uc3m | Universidad Carlos III de Madrid

Doctoral Thesis

Software Architecture for Self-Driving Navigation

Author:
Pablo Marín Plaza

Supervisors:
Dr. Arturo de la Escalera Hueso
Dr. David Martín Gómez

Electrical Engineering, Electronics and Automation

Leganés, July, 2018

Doctoral Thesis

Software Architecture for Self-Driving Navigation

Author: ***Pablo Marín Plaza***

Supervisor: ***Dr. Arturo de la Escalera Hueso***

Co-Supervisor: ***Dr. David Martín Gómez***

Signatures of the examination court:

President: _____

Spokesperson: _____

Secretary: _____

Leganés, July, 2018

. . . . - - - . - . . . . - . . - . . . . . - - - . . . - . . . .

Unfortunately, no one can be told what the Matrix is. You have to see it for yourself.

- Morpheus

Almost all accidents take place because of human distraction.

- Sebastian Thrun

To ask the right question is harder than to answer it.

- Georg Cantor

Success is not a good teacher, failure makes you humble.

- Shah Rukh Khan

Success is 10% talent and 90% hard work.

- Chinese proverb

For you and only you that are reading this work,
for you that are curious about the world and how the things work.

. - - . . . . - - . . . . . - - - .

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are my original work toward the degree of Doctor of Philosophy at Universidad Carlos III de Madrid. This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text.

Pablo Marín Plaza
July 2018

Acknowledgements

I would like to thank the support of the LSI team from the beginning and most especially this last year, in particular to my directors Arturo and David and additionally to José María and jefecito Fernando. My most sincere gratitude to Ahmed because without him, this project would not have worked and I would never have learned English. Thank you, Fran and Miguel, for helping in the last stages of the project. I am glad to count with my family and friends which have been there always to support me unconditionally. Finally to all the people interested in the success of this work.

Abstract

This dissertation is based on the development of the navigation software architecture for self-driving vehicles. The goal is very wide in terms of multidisciplinary fields over the different solutions provided, however, functional solutions for the implementation according to the software architecture has been proved and tested in the real research platform iCab (Intelligent Campus Automobile).

The problems that the autonomous vehicles have to face are based accordingly as the three questions of navigation that each vehicle has to ask: Where am I, where should I go, and how can I go there. These questions are followed by the corresponding modules to solve that are divided into localization, planning, mapping, perception and control in addition to multitasking allocation, communication and Human-Machine Interaction. One more module is the self-awareness which is an optimal solution for detecting problems in the earliest stage.

Throughout this document, the solution provided in form of a complete architecture for navigation describes the modules involved and the importance of software connections between them, generation of trajectories, mapping, localization and low level control. Finally, the results section describes scenarios and vehicle/software performance in terms of CPU for each module involved and the generation of trajectories, maps and control commands needed to move the vehicle from one point to another.

Resumen

Este documento es el resultado de cinco años de trabajo en el campo de los vehículos sin conductor donde, en él, se recoge el desarrollo de una arquitectura software de control para la navegación de este tipo de vehículos. El objetivo es muy ambicioso ya que para su desarrollo ha sido necesario el conocimiento de múltiples disciplinas como ingeniería electrónica, ingeniería informática, ingeniería de control, procesamiento de señales, mecánica y visión por computador. A pesar del basto conocimiento necesario para lograr un vehículo funcional, se han alcanzado soluciones para cada uno de los problemas en que consiste la navegación autónoma, generando un vehículo autogobernado que toma decisiones por sí mismo para evitar obstáculos y alcanzar los puntos de destino deseados.

Los problemas principales que los vehículos autónomos tienen que hacer frente, están basados en tres preguntas principales: dónde estoy, dónde tengo que ir y cómo voy. Para responder a estas tres preguntas se ha dividido la arquitectura en los módulos siguientes: localización, planificación, mapeado del entorno y control junto con módulos extra para dotar al sistema de más aptitudes y mejor funcionamiento como por ejemplo la comunicación entre vehículos, peatones e infraestructuras, la interación humano máquina, la gestión de tareas con múltiples vehículos o la propia consciencia del vehículo en cuanto a su estado de baterías, mantenimiento, sensores conectados o desconectados, etc.

A través de este documento, la solución proporcionada a cada uno de los módulos involucrados refleja la importancia de las conexiones de software y la comunicación entre procesos dentro de la arquitectura ya sea para la generación de trayectorias, la creación de los mapas a tiempo, la localización precisa en el entorno, o los comandos generados para gobernar el vehículo. Así mismo, en el apartado de resultados se pone de manifiesto la importancia de cumplir los plazos de compartición de mensajes y optimizar el sistema para no sobrecargar la CPU.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

In 2017, 1.25 million people suffer a premature death as a result of road traffic accidents. Most of the casualties affect people between 15 and 44 years old around the world. The risk factors are mainly humans as the result of over limit the speed, drowsiness, drugs, and distractions such as GPS screens manipulation, mobile phone writing while driving and while crossing the road as pedestrians [71]. Figure 1.1 shows the casualties per 100.000 inhabitants worldwide.

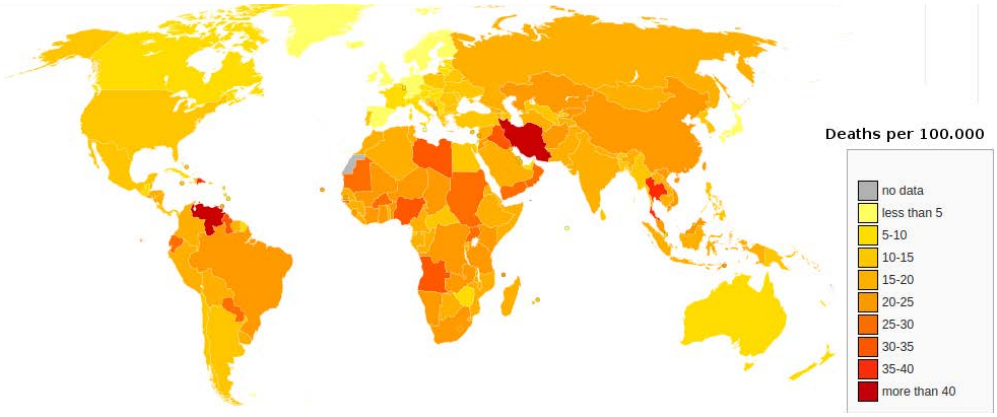


Fig. 1.1 Road traffic accidents world map.

The government of each country has to deal not only with the casualties on the roads but with the nonlethal accidents where people get injured or urban material is damaged. Part of the solution to this problem is under development from the past few years, where the transportation system involves intelligent vehicles and infrastructure to create smart cities. The main idea is to connect every vehicle to a network which provides useful information such as traffic lights, vehicle density or blocked streets among other useful information.

Nowadays, each family has one or two vehicles and the consequences for the city is the overpopulation of vehicles, pollution and traffic jams during specific hours. Consequently, in the publication *The End of Car Ownership* [33], Higgins remarks that in the future, no one will own a vehicle and shared vehicles as public services will be implemented in the cities. Furthermore, in the future, one worker could go from its house to the working place just requesting a vehicle to pick up in its house and go to work without traffic jam, delays or traffic accidents. The vehicles parked in the streets will be no longer a problem in the cities due to the reduction of vehicles in the city. Another publication such as "What will the car of 2040 be like?" in [27] explains that the companies nowadays are investing for a future with vehicles as a public service for a monthly amount instead owning the cars.

In smart cities, vehicles are intended to drive without human intervention in order to avoid human risk factors aforementioned linked to a manual driving. Hence, the first step in this challenge is the developing of platforms able to navigate in urban environments and highways. This is not an easy task despite it looks easy to an experienced driver. The research community of autonomous driverless vehicles is in the first stage as if they are novelty drivers, so, it is necessary to learn and evolve for the developers in order to create every day better and safety driverless vehicles.

Continuing the analogy, the rookie drivers requires a huge amount of brain activity in order to face new situations where the focus level is at its maximum expression. During the learning period, this driver develops some skills that reduce the activity in the brain, keeping the focus level at a high rate where it assumes some situations as normal and evaluates the possibilities in each intersection and negotiations. This same situation requires huge computational power for CPUs inside of the vehicles where they have to evaluate situations which a priory are ready to make decisions based on the perception of the environment. From last years, the rise of new technologies, more powerful computers and novelty algorithms impulses this driverless development to a new level.

Hence, the first step is the creation of reliable and functional vehicle able to navigate from one point to another by itself which is the main objective of this dissertation. Once the vehicle is able to navigate safely in the free space and making the correct decisions, the next goal is the communication between other vehicles, infrastructure, and pedestrians, which is the basis of controlling multiple vehicles. Furthermore, in a system with multiple agents, it is required the multi-task allocation which will decide what tasks are assigned to each agent, in this case, to each vehicle.

In summary, it is required a vehicle able to navigate around the environment with the capacity to make decisions by itself and avoid collisions. After the validation of this vehicle, it is necessary the communication layer to enhance the decisions and to cooperate with other

vehicles and infrastructure. Finally, the creation of a swarm of vehicles where there is a multi-task allocation for future vehicles as public services.

Several problems that the vehicles need to face nowadays are the entries and exits in highways, intersections, parking, traffic rules for each road by detecting and identifying traffic lights, traffic signs, road marks among the pedestrian interaction in pedestrian crossing places and the interaction with other human drivers in the transition stage.

The goal of this dissertation is the creation of a self-driving vehicle which is able to navigate from one point o another inside of a known environment as a first step for the future smart cities. Fields like mechanical engineering, informatics engineering, electronic engineering and robot science are required in order to develop from scratch the first steps into a functional research platform where several methods to solve problems related with the navigation will be tested and used. Furthermore, this dissertation is the description of the full software architecture for self-driving navigation that has been developed during this five years in order to obtain functional self-driving vehicles.

1.1 Motivation

Due to the factors aforementioned exposed, the main objective to this thesis is to create a research platform that allows to anticipate to the future smart cities and start solving problems about perception, planning, control, and navigation. For this task, the whole project has been tested in a modified electrical golf cart. Furthermore, one objective accomplished is the development of iCab project where a fleet of autonomous cabs navigates around the campus to pick up and drop off people with reduced mobility.

Around five years ago, the LSI offered me the opportunity to help with two electric golf cars modified with control devices for steering and traction movement which were not functional because the low-level software was not ready. Working the first year in the communication between the low-level micro-controller and the main computer was not an easy task due to the hardware problems but progress was made and at the mid of the second year, LSI owns two functional electric golf carts, manually driven by a joystick. The following year the setup of hardware devices such as sensors and actuators for the new software architecture were taking form and by the end of the fourth year, the iCab (Intelligent Campus Automobile) was in its final steps with high-level software architecture where perception, localization, trajectory generation and control work together to navigate from one point to another in the campus.

1.2 Main contributions

The main contributions listed below shows the parts where I was involved and working actively in the iCab project. All of them are explained in its correspondent chapter or are well referenced to published conferences and journals.

- Low-level software development between the main Linux computer and PIC microcontroller based on serial communication. I developed this software from scratch using C++, serial communication, marshaling structures for transmission between CPU and PIC and adding the heartbeat in order to detect abnormalities.
- Low-level speed control. This module is fully developed by me using PID adjusted to maintain the reference velocity. It is adjusted firstly the P gain, secondly the D gain and finally the I gain with windowed limitations for the integral part and derivative part.
- Localization module which involves encoder odometry and GPS odometry; the modification and integration of lidar odometry with covariance, visual odometry and the creation of odometry manager which select which odometry will be used from the available sources; the configuration of initial localization AMCL module.
- Perception module which involves the fusion of stereo camera and laser in order to detect steps or low height obstacles with a publication in a conference [59].
- Trajectory generation module which involves the integration, configuration and adaptation of *move_base* software from ROS repository described in chapter 5
- Graphic user Interface fully developed from scratch in order to control and manage the inputs and outputs of the vehicle making easy the debug and present a new friendly form of demonstrating the possibilities of each module into the system.
- Hardware and cable management which includes the emergency button to stop the vehicle, lights and warning sound installed in the frontal of the vehicle in order to make sure people in the campus are aware of the vehicle.
- Software architecture to manage all modules and the information shared to make the system coherence and reliable in addition of the configuration and development of all transformations between the static vehicle elements as sensors and the odometries.

During the last two years, I also work with ISIP40 research group from University of Genova (UniGe) (Italy), in the field of self-awareness state of the vehicle using neural networks and machine learning.

Chapter 2

State of the Art

Self-driving vehicles are designed using several scientific disciplines in order to solve navigation problems such as Localization, Perception, Mapping, Control, Human-Machine-Interaction (HMI), Decision Making among others, but none of them are useful if there is no coherence and synchronization between all the components involved. As a result of this interaction, considerable information provided for each module to the system needs to be managed such as environment information, self-status, and emergency events. This task corresponds to the software architecture, designed to link modules, accomplishing time limits and synchronizing vital information.

With the first DARPA Grand Challenge in 2004, different research groups notice the importance of having competitive software and hardware architecture by experience. Red Team from Carnegie Mellon, with the top score, was able to drive 12km (5.4%) in the Mojave Desert with the vehicle Sandstorm (a Humvee model) [90] equipped with lidar, laser, radar and stereo camera sensors. Among the rest of the participants, three of them were able to drive 8km, four teams were able to drive 1.5km and the seven last teams were not able to start the race. The price was two million American dollars granted by the Defense Advanced Research Projects Agency and nobody could reclaim it. After this failure performance, in 2005 another DARPA Grand Challenge [14] took place in southern Nevada. In this event, five teams fully completed the challenge of driving autonomously 212km. The winner was Stanford University with the vehicle Stanley [88] with the top score of 6 hours and 53 minutes claiming the price of two million American dollars. Challenge by the challenge, research groups and companies discover the potential of autonomous vehicles and in 2007 DARPA proposed the Urban Challenge to drive autonomously in urban environments. The whole setup was done in Victorville, California, and the participants had to make the vehicle to drive over a staged city environment negotiating other moving vehicles and obstacles while obeying traffic regulations. In this occasion, Tartan Racing from Carnegie Mellon with the vehicle

Boss [20] was the winner claiming two million American dollars as a price. After the events,

Fig. 2.1 Vislab vehicle.

researchers, companies, and governments around the world understood the importance of self-driving vehicles for society and the dream started becoming a reality. Some examples of the advances in autonomous driving vehicles were reaching by Vislab group, in Italy, which drove 13.000km from Parma to Shanghai in 100 days during the summer of 2010 [9]. The fleet of four vehicles performed the whole trajectory without human intervention. In the same year, Google announced the accumulation of 225.000 Km with its Toyota Prius and Audi TT and later with a fleet of 23 Lexus RX450H SUV [29] reaching more than one million of kilometers by 2014. Another example was in 2014 when Mercedes [98] drove Bertha, an autonomous vehicle, over 103 Km with rural roads, small villages, and cities using perception localization, digital maps, and complex planners.

Fig. 2.2 Bertha vehicle.

Along the evolution of this groups, several research projects fulfill the streets with autonomous vehicles for testing purposes such as CityMobil [76] and CityMobil 2 [1] in France; GATEway, UK Autodrive or VENTURER in United Kingdom [16]; Waymo by Alphabet, the spin-off of Google; Sedric [13] in Shanghai; nuTonomy in Singapore in 2016 and nowadays, the list of private companies working in Intelligent transportation and manufacturers interested in autonomous vehicles is endless.

All the vehicles from these projects are designed to navigate from a starting point to a goal point avoiding collisions and taking its own decisions. In order to classify the vehicles by the automated performance, table 2.1 explains the six levels of autonomy defined by Society of Automotive Engineers [81]. From the zero level where the vehicle is fully driven by the human, to the fifth level where the needed operations are managed by the automated driven system.

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	No Automation	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	Conditional Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	High Automation	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	Full Automation	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Table 2.1 Autonomy levels.

Nowadays, prototype vehicles are able to reach level four, while commercial vehicles under development are able to reach level three top due to regulations and insurances. There are three main companies leading the race for the commercial autonomous vehicles: Tesla Company, led by Elon Musk, with the Autopilot version 8.1 and the Hardware 2.0 installed in the most recent vehicles, which affirms that they will reach the fifth level in 2018. Uber company, led by Dara Khosrowshahi, will release a fleet of self-driving cabs by around mid-2019. Waymo company, led by John Krafcik, uses the vehicle 600 Chrysler Pacifica minivan for all the fleet and focus its efforts on making these vehicles drive to the places

where people normally go such as work or school intended to learn habits from normal people.

Fig. 2.3 Tesla.

Fig. 2.4 Uber.

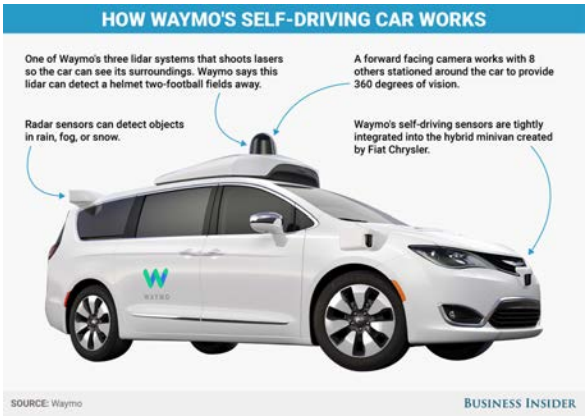


Fig. 2.5 Waymo.

2.1 Software architecture for self-driving vehicles

As said before, one of the most important aspects in autonomous vehicles is the design of software architecture, to manage the information provided by the physical devices and act accordingly. This architecture is highly dependent on hardware connections and performance (or role) of each CPU or device mounted in the platform, hence, it is possible to classify the architecture by connections and by performance.

2.1.1 Hardware connection

For the connections point of view, there are three approaches in order to gather and process sensor information and forward it to actuators: Centralized, Decentralized and Distributed. This classification is based on the connection between sensors, CPUs, and actuators. It is mandatory to know the objective and goal for the vehicle which is in development before selecting the connection distribution. There is a slight analogy between the network architectures and hardware architecture inside of an autonomous vehicle. This fact is in part of the huge amount of information which the vehicle needs to manage to take the best possible decision.

Centralized

In this architecture, the information flows directly to the main CPU which will evaluate, process and estimate the best trajectory. The sensors could be connected to this main CPU or other auxiliaries CPUs but the actuator is directly connected to the main CPU, hence, there is only one responsible of the movement. This topology is the easiest and fastest one to develop but the runtime processes to check the self-state of hardware and software gives all control to only one CPU, therefore, one node failure and the vehicle will become very dangerous. Figure 2.6 shows the connections between sensors, CPUs, and actuators in the vehicle.

Decentralized

In this architecture, there is still one main CPU taking decisions but other auxiliaries CPUs are connected to the actuators in order to act when required. This hardware architecture is perfect for Emergency situations when the main CPU is not able to achieve a solution or there is an emergency event. The information flows to the main CPU but other CPUs are checking the health of the system or the self-state of the vehicle but more importantly, this auxiliaries CPUs can performance small routines such as keep in the lane or change lane, automatic parking or control the velocity and steering angle among others. The development stage of

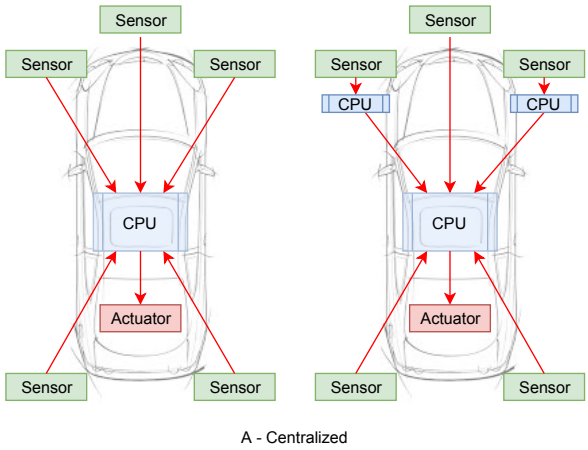


Fig. 2.6 Centralized hardware architecture based on its connections.

this hardware architecture is larger than the centralized and requires to think further into the safety aspects of hardware, software, and performance. Figure 2.7 shows the connections between sensors, CPUs, and actuators for this architecture.

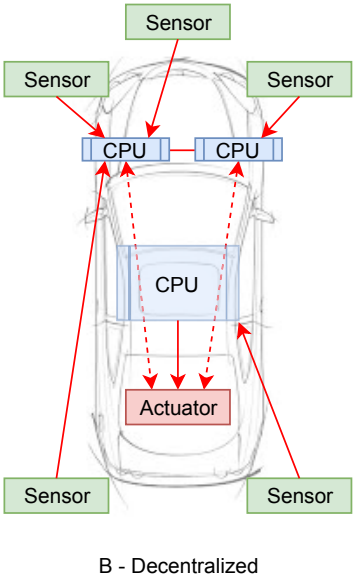


Fig. 2.7 Decentralized hardware architecture based on its connections.

Distributed

The last one is the distributed connection devices which will every CPU is connected to a bus and could access the information of every sensor and can act in every actuator. The main advantage is the easy access to sensor information in each CPU but the drawback is limited

bandwidth, one example is CANBUS. Figure 2.8 shows the connections between sensors, CPUs, and actuators by the data BUS where every CPU have access to sensor information and actuators.

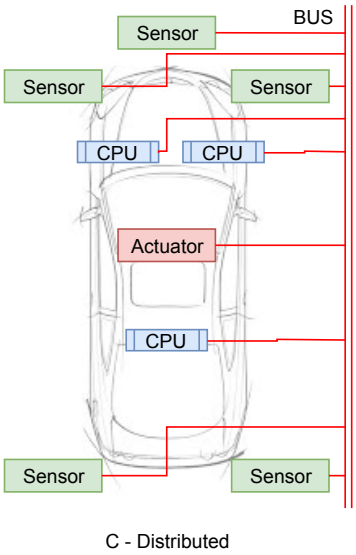


Fig. 2.8 Distributed hardware architecture based on its connections.

2.1.2 Software connection

This classification is highly related to the software used in the vehicle which will manage the information and will be the most critical part of the decision making. As is described in the next chapters, the software architecture is divided into modules which will solve one or several problems related to the autonomous vehicle. These modules are running by programs (nodes) which are interconnected sharing information and there is another similar analogy to the network topology in the classification.

Centralized

Figure 2.9 shows the connections between nodes, sensors and actuators and only one node which gather, process and evaluate the solutions. This architecture is easy to develop for small and simple projects because is easy to track the variables and functions, there are no sharing information between process and everything is synchronized and instantaneous but it is necessary to keep in mind the cycle rate to accomplish the task. This architecture is not optimized for autonomous vehicles due to the huge amount of data and problems to solve. Moreover, the management of everything in one node and keep the track of each variable is almost impossible.

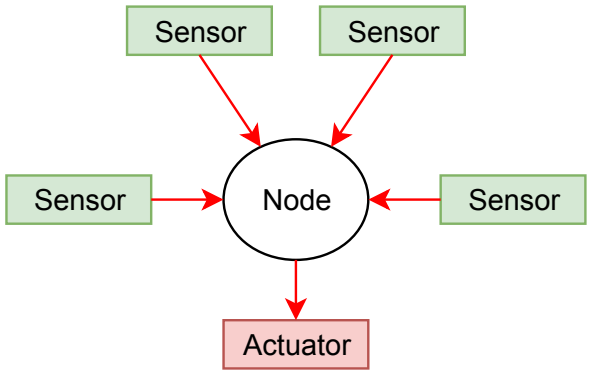


Fig. 2.9 Classification of architectures based on software connections.

Decentralized

Figure 2.10 shows the node configuration where sensors and actuators are outside of the core of the architecture providing a HAL (Hardware Abstraction Layer) which its main advantage is to isolate the physical device to the architecture making it possible to change the sensor easily without modifying the nodes inside of the architecture. The elements inside of the architecture are connected to each other generating high-level information which flows upstream up to the decision making level. It is very useful for research projects where new techniques and devices are in constant evolution.

The solution provided for each classification is valid as long as the computational time and the bandwidth allow communication between processes. Along with different features for each type, the main factors to take into account in order to select what system is suitable for the autonomous vehicles are Maintenance, Points of Failure, Fault Tolerance/Stability, Scalability, Creation/Evolution, Backwards Compatibility, Security.

- Maintenance
  This element is crucial for long-term projects. When time goes by and the system needs software and hardware updates and new features are added, multiple connection points need to be checked if the new changes affect other modules. Centralized systems are easier to maintain than distributed.

- Points of Failure
  The chain of connections between processes that exchange messages is easy to track in centralized systems as there is only a single point of failure, all the processes are

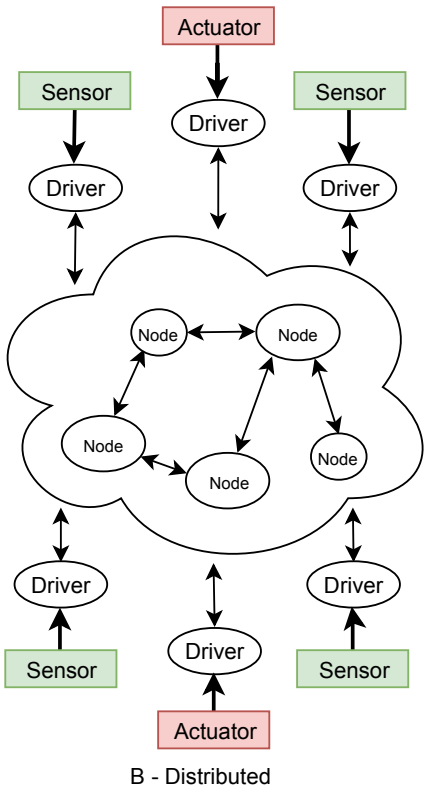


Fig. 2.10 Classification of architectures based on software connections.

running in the same computer. Distributed systems have more points of failure and tracking error are less intuitive but still traceable.

- Fault Tolerance/Stability
  The definition of fault tolerance is the ability of the system to continue working when a program fails. For centralized systems, one error in the node is crucial because there is not turn back or recovery. As one program fail or stuck, the whole system is highly unstable and the vehicle becomes dangerous. For distributed systems, if one program dies, other nodes work in parallel to fail-safe the system.

- Scalability
  This factor is really important in the design stage due to the future feature additions to the system. For example in the case of the autonomous vehicles, if new traffic rules emerge, the codification in centralized systems implies to redesign the main logic. In distributed architectures, changing the module in charge of the traffic rules it would be enough to integer the new feature.

- Creation/Evolution
  The easiest and fastest architecture design is the centralized due to the fact that everything is connected and the effort on the connection and synchronization is neglected. In distributed architectures, the connection manager and synchronization layer have to be taken into account from the beginning in order to create a healthy framework to share information.

- Backward Compatibility
  This feature allows the architecture to work with downgrade versions of modules or parts of itself. Each type will depend on the framework used.

2.1.3 Safety standards and cybersecurity

The Federal Motor Vehicle Safety Standards (FMVSS) and the National Highway Traffic Safety Administration (NHTSA) have worked together in order to create regulations for autonomous vehicles including cybersecurity. Regulating this cybersecurity is a serious challenge in elements that put on risk lives and infrastructure. Due to the fast advance in technology and low speed on regulation, these drafts are not complete or obsolete and by the time of this dissertation, a good regulation in this matter is still missing. Some extract of the draft for the requirements published in 2015 from the California Department of Motor Vehicles [17] is:

"*Privacy and Cyber-Security Requirements Manufacturers will provide a written disclosure to autonomous vehicle operators of any information collected by the autonomous technology that is not necessary for the safe operation of the vehicle and will be required to obtain written approval to collect this information. Autonomous vehicles will be equipped with self-diagnostic capabilities that meet industry best practices and are capable of detecting, responding and alerting the operator to cyber-attacks or other unauthorized intrusions. In the event of such an alert, the autonomous vehicle operator will have the capability to override the autonomous technology.*"

2.1.4 Tasks

The evolution of the tasks fully related to the vehicle has been changed over the time. At the beginning of improvement assistance driver, the inner tasks for the vehicle were merely throttle, brake and steering wheel actuators. The emerge of Driver Assistance such as power steering, cruise control, parking assists, the anti-blocking system (ABS), electronic stability control (ESP) highly facilitates the manual driving. One step further was the Advance Driver

Assistance with adaptive cruise control (ACC), autonomous parking, collision warning, lane keeping assists, advance emergency brake (AEB). The next generation of tasks belongs directly to the autonomous vehicles with no human intervention. The figure 2.11 shows the implementation of these tasks over the years [6]. Furthermore, in the work of Sagar

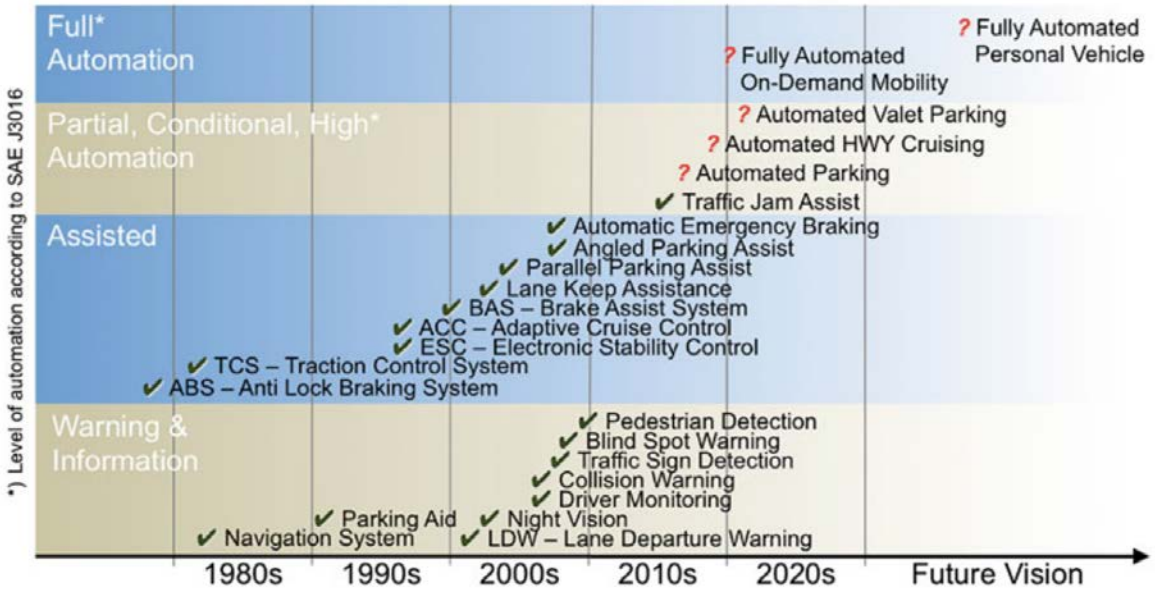


Fig. 2.11 ADAS Timeline from [6].

Behere [5], he splits the components of autonomous navigation into layers and discriminates what of this components belong to the vehicle platform and which ones to the cognitive driving intelligence. Regarding this classification, the sub-figure 2.12a represents one of the bases of autonomous navigation system architecture where the vehicle is able to handle only the actuators. In sub-figure 2.12b, the vehicle is in charge of its state as the self-energy management and diagnostics in addition to trajectory execution, self-stabilization, and actuators. Tasks as localization and sensor fusion are still fragmented and not fully linked to the vehicle which acts only as a black box for throttle and brake. The third paradigm in autonomous vehicle architecture is displayed in sub-figure 2.12c, which shows the high-level architecture developed with a cognitive layer to analyze the environment and making decisions by itself.

With this approach, there are multiple and very similar solutions. Some examples of the software architecture for autonomous vehicles are in the Mercedes Benz "Bertha" in figure 2.13 or ZMP Robocar in figure 2.14 based on Toyota Prius.

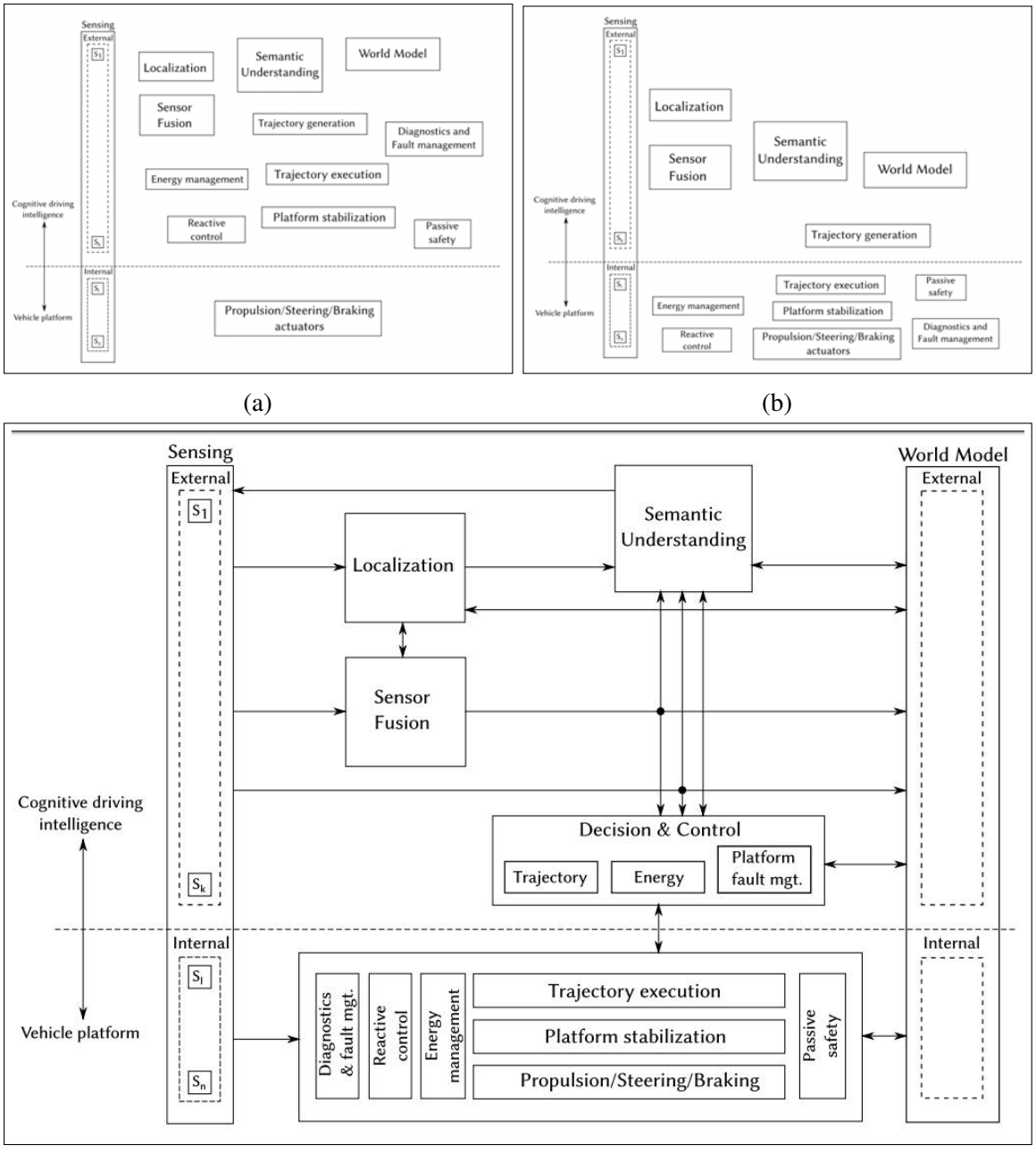


Fig. 2.12 Classification of architectures by tasks [5].

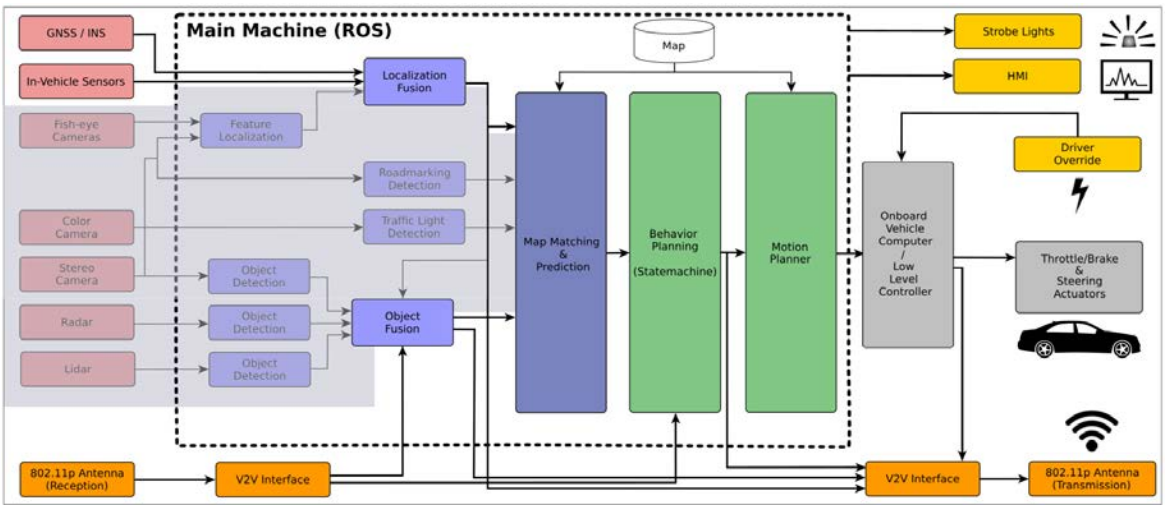


Fig. 2.13 Mercedes Benz S-class architecture [98].

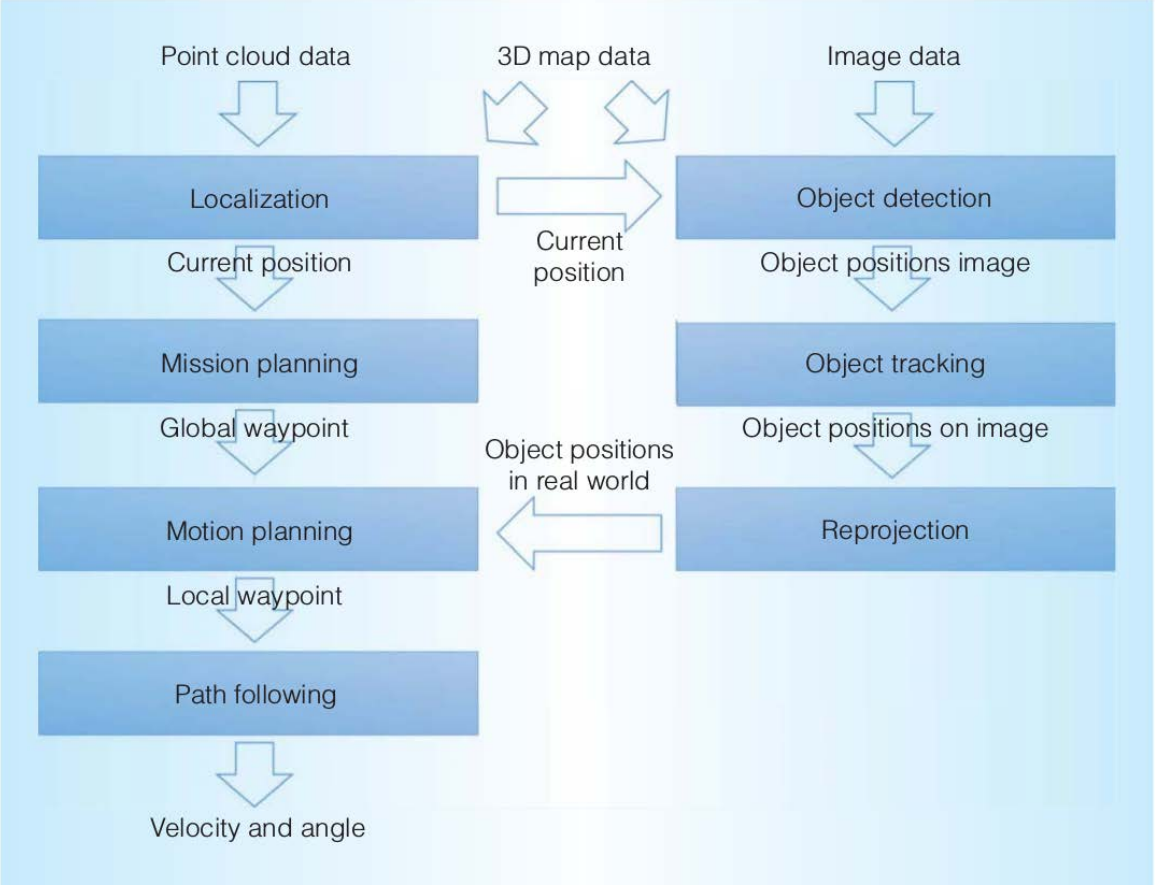


Fig. 2.14 ZMP robocar architecture [45].

2.2 Sensors

The devices that allow the measurement of the environment to extract vital information for navigation are the sensors. Each of them provides different information in type and form and all of them requires a driver to convert this information by using digital information for the system. The following list explains the use of each sensor into the architecture:

- Laser provides the measurement of the environment based on the time of flight of the light. The commercial devices have a wide range of models that vary the field of view and number of measurements. This device is good for mapping and obstacle detection among other applications.
- Radar provides obstacle information based on sound. The applications are detecting the type of road or type of obstacles [34].
- Camera provides images of the environment. Normally they are located around the vehicle to "see" what is in the surroundings. The applications are for obstacle identification, optical flow or tracking.
- Stereo camera is an advanced technology that provides not only the information of the monocular camera but it provides depth information. With this device is possible to estimate the odometry and obstacle detection.
- GPS receiver is a device used for localization and odometry. The accuracy depends on the sensor and the free space between the satellites and the device. Normally, this sensor is fused with another source of odometry due to the low accuracy.
- IMU is an inertial measurement unit sensor that provides the accelerations and angular velocity for each ax. It is useful in combination with GPS and a very good source of information for the dynamic model.
- Ultrasonic sensor is based on time of flight. The emitter sends a pulse that bounces to the obstacles and returns to the receiver providing the distance to the obstacle in range. It is good and not very expensive to cover the dead zones around the vehicle in order to complete the map.

2.3 Software prototyping tool

Aside from the architecture configuration, it is necessary to introduce the software framework from which all responsibility depends. From the Operative System to the prototyping tools,

it is necessary to define the ways of communication, controllers, computers, IDEs (Integrated Development Environment), debugging stage and simulation capabilities for the vehicle in order to create a functional architecture for an autonomous vehicle. All these elements are vital in the first stage of the design because the problems will be solved according to the specifications.

2.3.1 RTMaps

This tool creates the whole environment for processes communication between software running in parallel. It supports different languages like C/C++, C#, Java, Python, and QML. There are different interfaces and drivers for the most common sensors used in the automotive field like Cameras (FireWire, Ethernet, USB), Stereovision, CAN, LIN, FlexRay, XCP, GPS (IMU), Radars, LiDars, Ultrasonics, Biometrics, Data acquisition, Image Processing (OpenCV). They have support for the compatibility [79] of simulators like ProSIVIC, PreSCAN, MotionDesk, SCANeR and the last actualization they incorporate a compatibility with another framework like ROS. The use of this software is easy and simple. It has a graphical interface (figure 2.15) where it is possible to connect the nodes manually taking into account the type of the messages. It has debugging support, playback support,

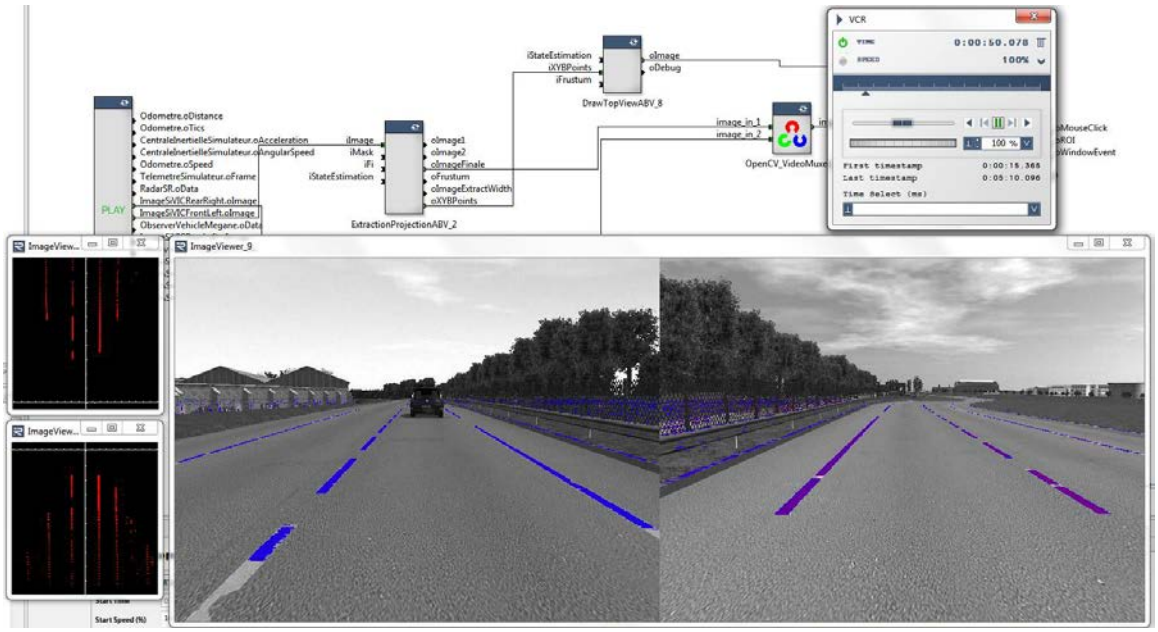


Fig. 2.15 RTMaps framework.

and synchronization support. Also works on Windows and Linux.

2.3.2 ROS

Robotic Operating System despite its name, it is not a full Operative system, this framework has the tools necessary to manage different software in centralized/decentralized or distributed systems. The layer of communication and synchronization between processes are transparent to the users allowing all the effort of the project focused on the main objective. The communication between processes follow the standard protocols TCP/IP sockets 2.16 and is compatible with multiple commercial devices. This tool is very similar to the other platforms aforementioned but has more extended and active community (numbers like over 1,500 participants on the ros-users mailing list, more than 3,300 users on the collaborative documentation wiki). There are several groups working on the standardization of messages for compatibility between systems. The core of ROS is licensed under the standard three-clause BSD license. This is a very permissive open license that allows for reuse in commercial and closed source products. The tools for software prototyping are wide and it includes debugging, IDEs and simulator. ROS works on Linux and MAC and supports languages like C/C++, Python, and Lisp. Exists experimental libraries for Java and Lua but not officially supported. Other interesting features are the possibility to work with other Frameworks with several parsers in order to communicate information inter-framework from different platforms. Despite the graphical tools, the connection between processes are not manually configurable and all the nodes should be configured before run them. One of the best features is the modularity provided by the communication logic: Each node can publish messages under a unique namespace. Other nodes configured with this namespace are able to listen. This abstraction of communication allows replacing one module for other really easy and with minimum changes. The visualizer is Rviz 2.17 and the official simulator 2D is Player/Stage and 3D is Gazebo.

2.3.3 YARP

Yet Another Robot Platform [95] is a framework (figure 2.18) designed for robots where it has a bunch of tools specifically to solve some of the most common problems regarding the robot world such as localization, navigation, obstacle detection, monitoring, communication among others. The framework is compatible with Linux and MAC and has a flexible interface with hardware devices and it is intended to extend the longevity of robot software projects. The basic communication protocols are TCP, UDP, multicast, local, MPI, mjpg-over-HTTP, XML/RPC, tcpros that can match other nodes working with different frameworks. Its libraries are written in C++ but with an extension, it is possible to run programs in python, java, ruby or C#. YARP does not have its own simulator but it is compatible with Gazebo which is one

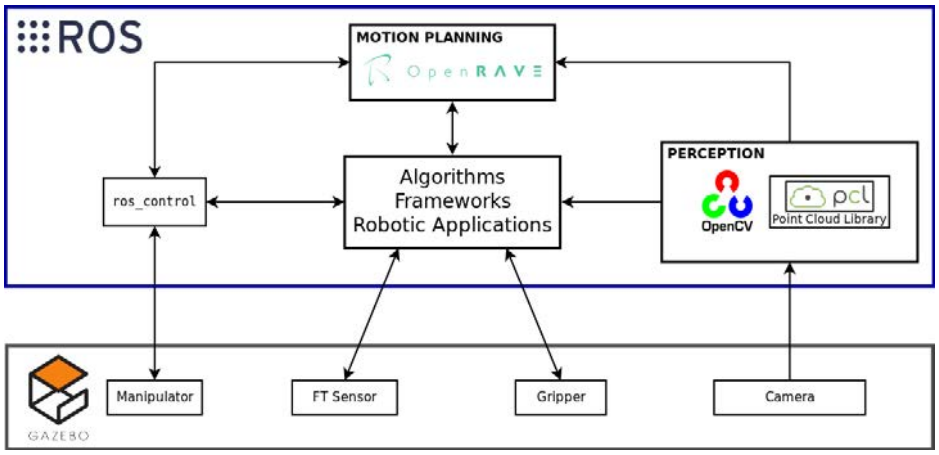


Fig. 2.16 ROS framework + Gazebo + PCL.

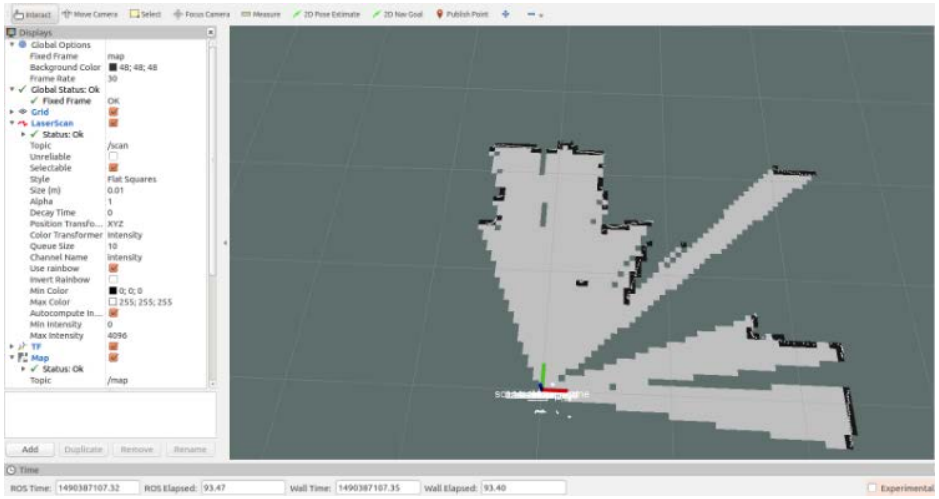


Fig. 2.17 Rviz Simulator.

of the most used in the field. It is very similar to ROS but lighter and it is intended to work with limited resources.

2.3.4 PAPCUS

PACPUS is a framework used in CNRS and Université de Technologie de Compiègne very similar to ROS and based on components, mainly written in C++. It was developed before the emerge of ROS and the developers chose to continue developing it with the license CECILL-C. Their "keep it simple" philosophy allows the user to communicate between components using pluggings, timestamps based synchronization, data recording and replay and advance tools for filtering, vehicle communication, etc. They have free to use their interfaces with the sensors, the HMI and viewer tools 2.19.

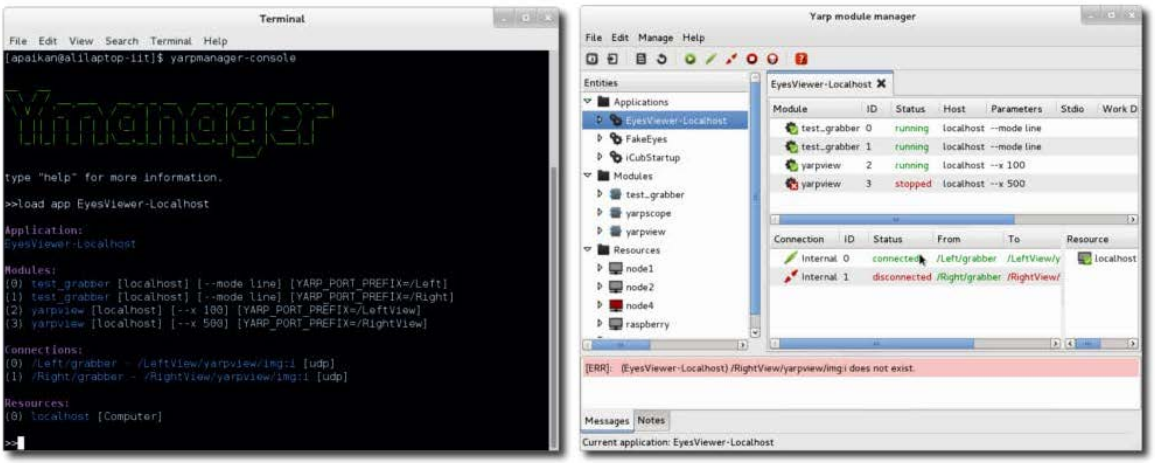


Fig. 2.18 YARP framework.

2.4 Modules

In order to solve the navigation problems, it is necessary to clear out what these problems are. As human beings, we interact with the environment with our legs, arms, and hands that act as the actuators. Our perception of the environment is reduced to our senses included the eyes, ears, skin, taste, and smell. But one of the important parts inside of us is the memory that store the events previously experienced and our inner sense of accelerations, normally forgotten. Our senses are able to close the loop to our brain that takes the decision is based on the complete or partial information. This main loop is close to the OODA loop described by Colonel John Boyd in the fifties (Observe, Orient, Decide and Act) The better sensors and actuators are, the better decisions one human being is able to make. This premise is extrapolated to the autonomous vehicles, as the better perception of the environment, better decisions the vehicle is able to make. First of all, one human being has to face three main questions for the navigation problem:

- Where am I?
- Where should I go?
- How can I go there?

The first question is related to the localization module. The second question is related to the planner module y and the last one corresponds with the control module. As human beings, we base our decisions on using auxiliary information like perception, obstacle identification, and memory to navigate into the environment. The vehicles need this auxiliary information for generating new modules that provide richer information.

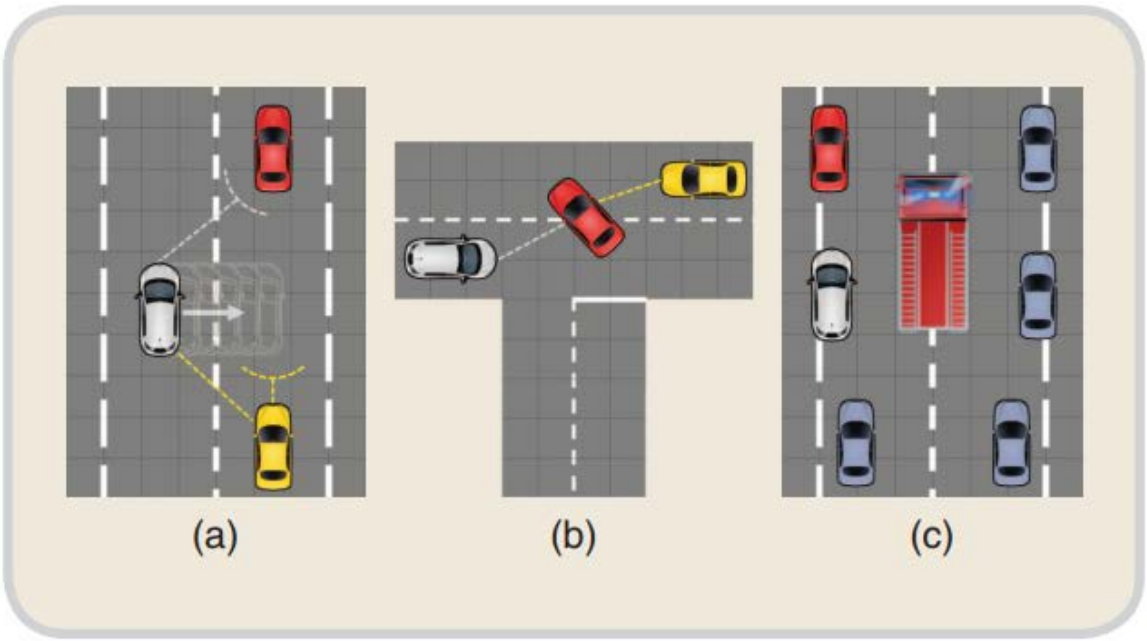


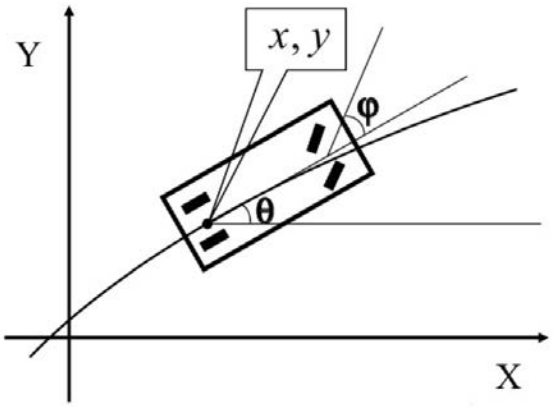
Fig. 2.19 Pacpus HMI over three different scenarios.

2.4.1 Localization

This module is in charge to locate the vehicle into the environment and translate the information of the real movement into digital movement. The measurement of how much the vehicle has moved (odometry) is not an easy task because multiple factors affect to the readings such as the mathematical vehicle model or sensor accuracy. The most common and simple method to extract the movement of the vehicle is using encoders sensors in the wheels and analytically extract the odometry using the Ackermann equations. Another technique based in different sensors such as visual odometry using stereo or monocular cameras, lidar odometry using Point Clouds, GPS odometry or WiFi footprints based on the WiFi signals around the campus. A review of different methods to obtain the odometry are exposed in this section with the description of the weaknesses and advantages of each technique.

Encoder/Wheel odometry

It is the simplest odometry technique. The odometry is composed using the encoder values from the direction wheels and the traction wheels. This approach uses the bicycle Ackermann parametric model 2.20 and the kinematic model for the localization as following where the v is the linear velocity, θ is the header angle, ϕ is the steering angle and the *wheelbase* is the distance between the frontal and rear wheel axis.



$$\dot{x}(t) = v * cos(\theta) \tag{2.1}$$

$$\dot{y}(t) = v * sin(\theta) \tag{2.2}$$

$$\dot{\theta}(t) = \frac{v * tan(\phi(t))}{wheelbase} \tag{2.3}$$

Fig. 2.20 Bicycle Ackermann model [92].

The movement is accumulated, so the error too. This is the less accurate method for the odometry generation because there are errors unavoidable like the sliding of the wheels, the pressure of the tires or the inaccuracy of the encoders. Work

Visual odometry

This method extracts the movement of the vehicle using images. It could be monocular odometry or stereo odometry:

- Monocular.
  The analysis of the images are based on Optical Flow [3] detecting features in the image and extract the translation and rotation matrix of the same features in the next image. Due to the use of consecutive images, the odometry is calculated by adding the new movement to the previous accumulated value, hence, the calculation must be perfect. The weakness of this method as said before, is the accumulated error. Some pre-processing and filters such as RANSAC or Kalman Filter [82] improves the outcome. New advances from the use of Optical Flow with optimized looking for the pixel displacement in the work [69]. Another approach based on images is the detection of landmarks in the environment knowing the location on the map. By geometry analysis, it is possible to calculate the position between the vehicle and the landmark and later, the pose of the vehicle into the map as [54], [47] and [51] suggest and the work of [30] test in the field.

- Stereo.
  This method uses a pair of images to have the extra information of the depth measurement. Different approaches using the UV-disparity [67] or Point Clouds(PC) [21] are

used combined with RANSAC or ICP to improve the results. Another approach is in the work of [80] where they promise the visual odometry based in stereo cameras that work in well-light outdoor environments and in dark indoor environments.

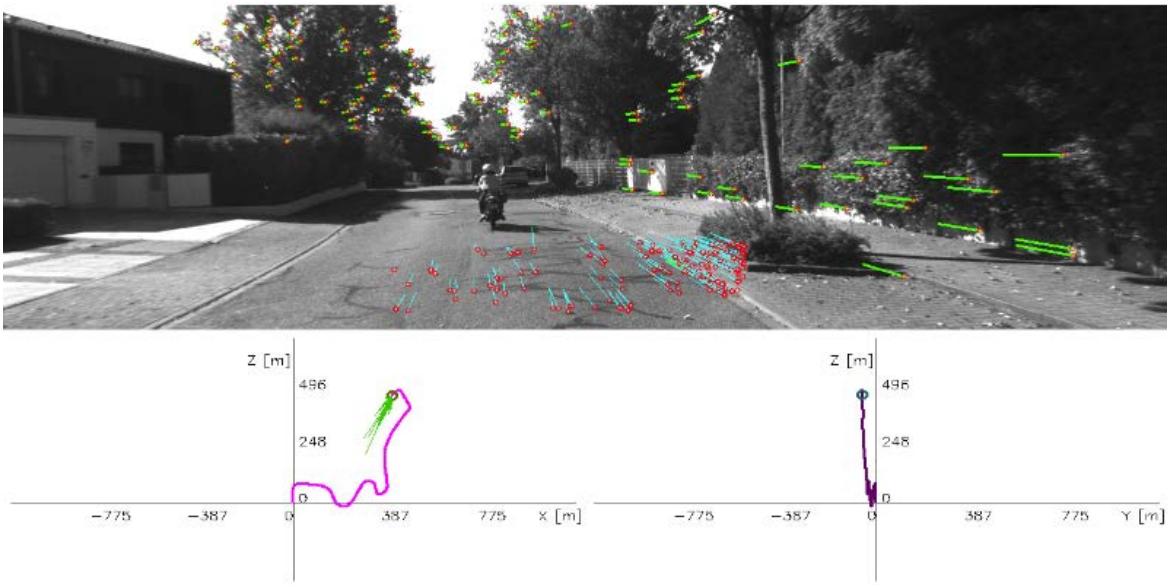


Fig. 2.21 Visual odometry using Optical Flow.

A complete state of the art about visual odometry is in [68] where several techniques are compared showing the strong and week points in monocular and stereo odometry approach.

Laser/Lidar odometry

This section is divided into 2D and 3D laser scans.

- 2D reconstruction of the movement. The principal idea is to use the data provided by the single layer laser to create a 2D map with the shape of the environment. After the movement of the vehicle, this shape is compared and correlated to extract the translation and rotation matrix and, again, it will be accumulated to the previous odometry. One example of this work combined with particle filters could be found in the AMCL (Adaptative Monte Carlo Localization) [86]. The AMCL method is based on accumulative movement in addition to particle filter for localization, hence, this method reset the accumulate error matching the localization in the global map, being less sensitive to odometry errors.

- 3D reconstruction of the movement. Due to the points generated in 3D (figure 2.22), with the lidar sensor, the points correlation is done with more accuracy but the computational cost is higher. Among other, the LOAM (Laser Odometry and Mapping) [96]

is a good example of how to use the ICP (Iterative Closest Points) [35] algorithm to match the points. Improving the previous LOAM, the work on [49] includes an inertial sensor to compute all the 6DoF (Degrees of Freedom).

One common approach is to combine and fuse the data from laser/lidar with images improving the results at expense of CPU and GPU computation. Work in [94] combines the lidar output (LOAM) with the VISO2 (visual odometry library) in order to improve both odometries. Work in [97] improves again both odometries making the final outcome more robust.

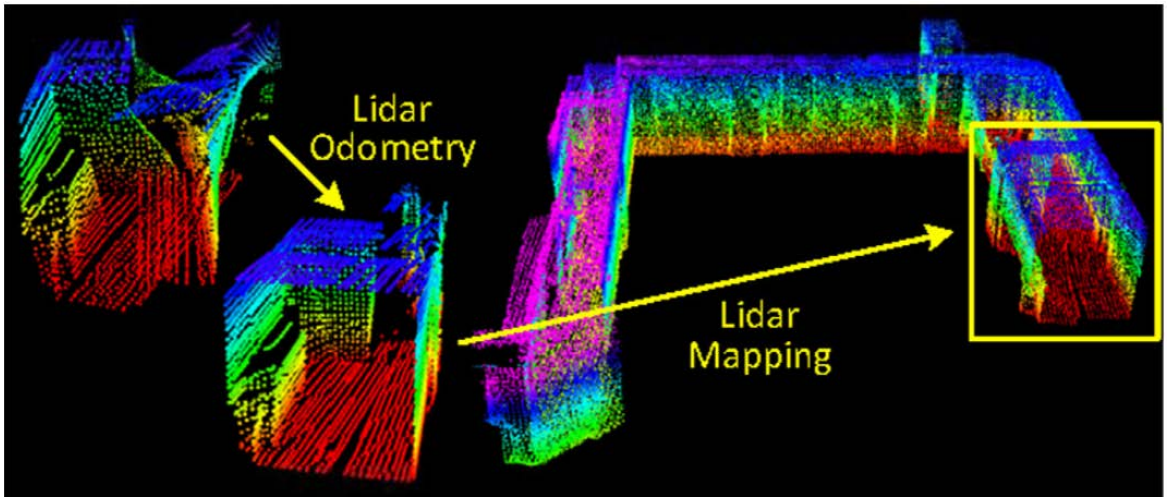


Fig. 2.22 Lidar odometry based on PCL matching points.

GPS odometry

This method is combined with the IMU (Inertial Measurement Unit) to increase the accuracy of the localization. It is based into geo-localization and, so forth, it has not an accumulative error but it is highly sensitive to the quality of the signal ($\pm 15m$). Some Differential GPS (DGPS) reach the accuracy of $\pm 0.1m$). Even with high precision DGPS, the readings could be shifted from the exact location, thus, it is combined with Kalman Filters and fusion information from other sensors. With differential RTK-GPS (Real Time Kinematic), the work in [64] promise high accuracy in low-cost device for environments where there is a quality signal.

WiFi odometry

Another localization approach based on WiFi fingerprints from the access points (AC) has been presented in [32] or [25] where the full map is divided into sections and for each section,

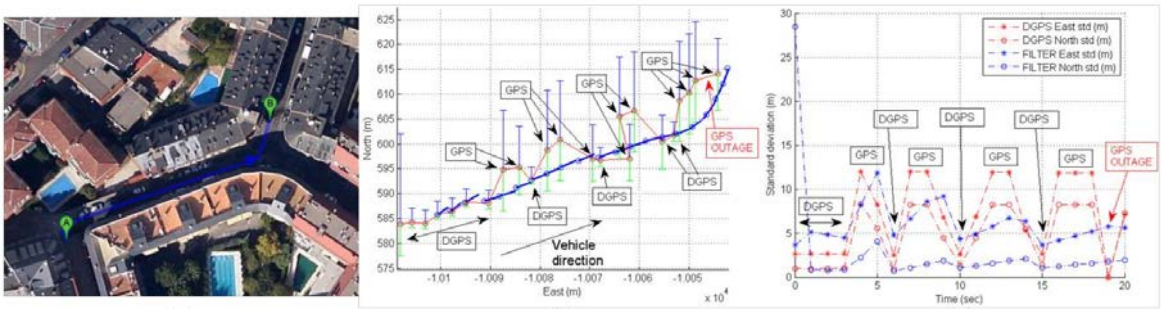


Fig. 2.23 DGPS odometry in combination of IMU with Kalman filter.

several parameters in the WiFi network from all AC is analyzed providing a localization into the map. Other methods are presented in [10] only for indoor navigation and some of them are combined with GPS in order to work outdoor near buildings [83]. The use of WiFi odomertry, figure 2.24 is intended to work for indoor environments but with the introduction of IoT devices and the infrastructure ready, this odometry suits perfect for smart cities.

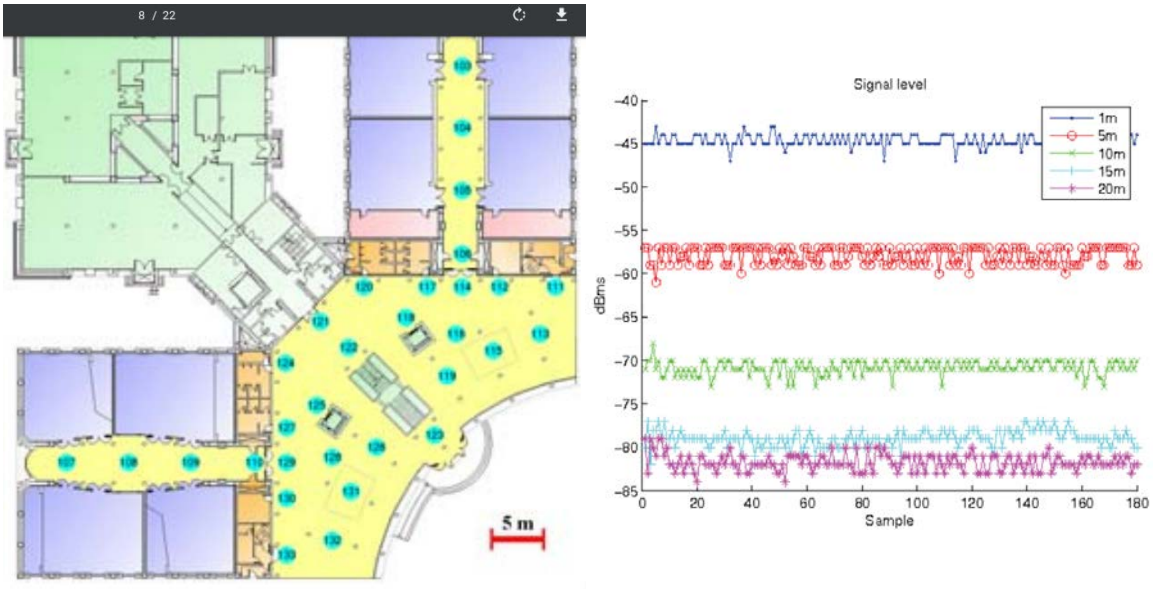


Fig. 2.24 Localization by WiFi.

Initial localization

This submodule is vital for the vehicle to understand the environment and localize itself in the digital map. Because all the movements are relative from one point to other, it is necessary to define the fixed point in the digital world to measure the global movement of the vehicle. Once the fixed point is defined, the vehicle should know where is its initial position using

this submodule. Some of the algorithms in the literature are based in EKF (Extended Kalman Filter) to match some beacons, corners or features reading by the sensors with the map and apply a Particle Filter to generate the initial localization as in [55] or [54].

This submodule is really important for global localization due to the reset of the error generated from the aforementioned methods. Do not matter how accurate the odometry is, with the movement, the vehicle will acquire some minimal deviances which will affect the digital position of the vehicle. Due to this deviance, some methods such as AMCL (Adaptative Monte Carlo Localization) analyze the sensor data to match with a known map and reset the vehicle position in this map [87]. Other methods are based on landmarks, figure 2.25, in order to triangulate the knowing point of the detected landmark with the camera installed on the robot and reset the odometry error [43].

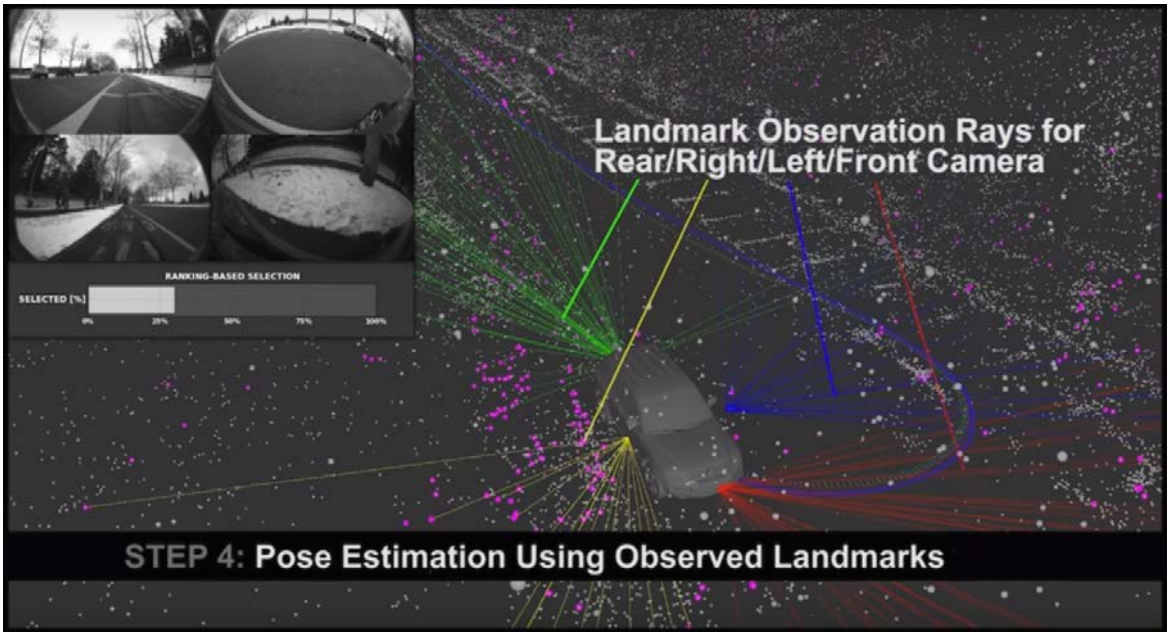


Fig. 2.25 Localization and pose estimation by landmarks.

2.4.2 Perception

This module is in charge to provide vital information to the system in order to generate appropriate maps, localization, and knowledge of the environment to improve the decision layer. Some literature includes all submodules into the perception layer and environment understanding but for better analysis, even knowing the direct relations from the other modules, in this work will be separated.

The same way human beings use their eyes, and ears in order to walk or drive, autonomous vehicles use their sensors to generate information. This information is under the supervision of the Perception layer and is used to detect and recognize elements in the environment to understand better the place of the vehicle in the real car.

Obstacle detection

The first step into perceiving the environment is the discretization of the static elements such as road, sky, buildings, etc. to the dynamic obstacles such as vehicles, pedestrians, traffic signs, lanes, etc. There are multiple approaches [24] to separate dynamic obstacles from the static obstacles using monocular vision based on the color space, edge detection or textures [89]; or using stereo vision, figure 2.26, dividing the image into free space navigation and obstacles by the UV-disparity map [50]. Another approach that has been rising over the past years is by analyzing the 3D information dividing the obstacles by the proximity of the clusters in the Point Cloud generated by the lidar or stereo camera [7], figure 2.27

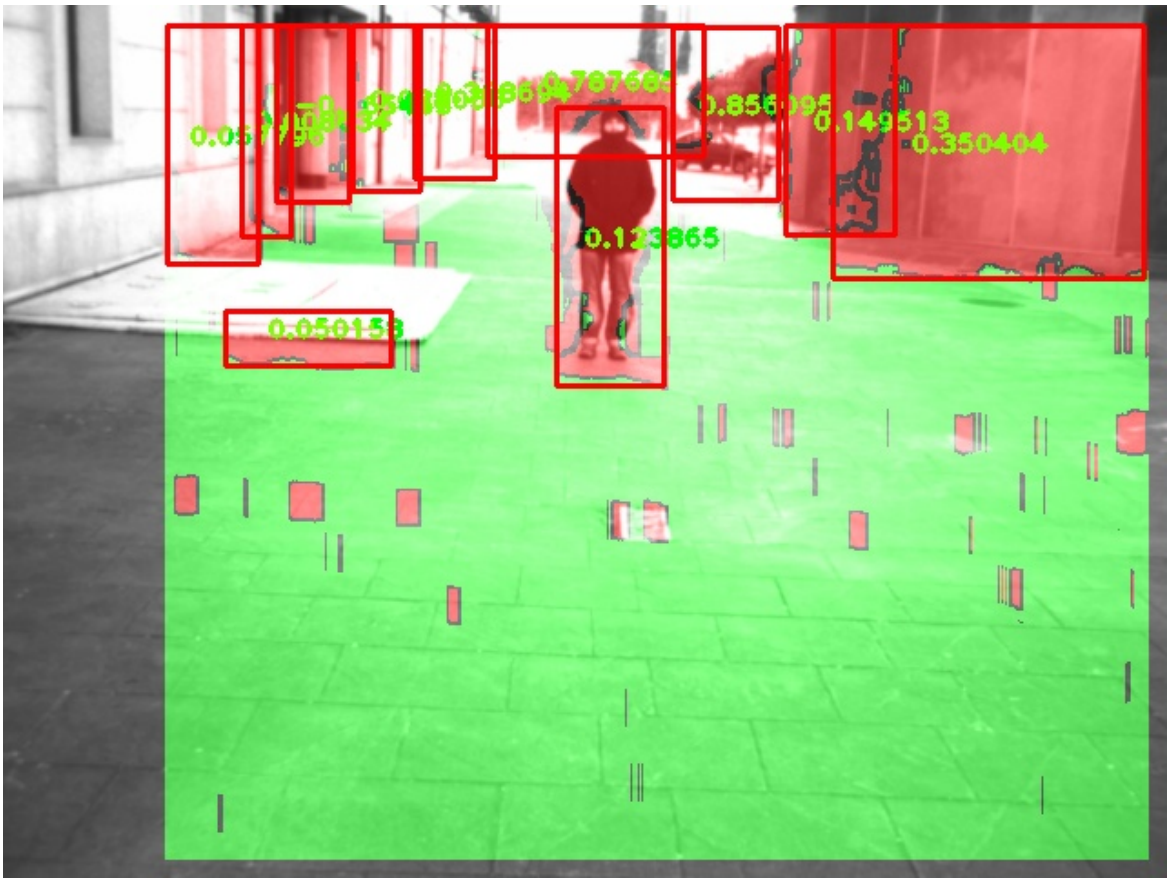


Fig. 2.26 Obstacle detection by stereo camera.

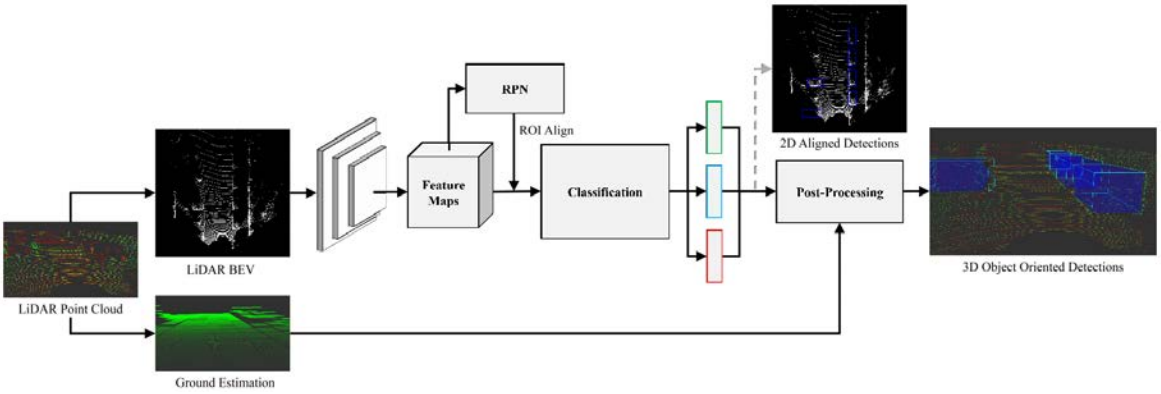


Fig. 2.27 BirdNet obstacle oriented detection.

Obstacle recognition

Once the obstacles are separated from the background, it is time to label them by categories. These categories will divide the information according to the task, for example, the vehicle obstacles will be treated differently as bicycles or pedestrian. The approach to distinguish between obstacles has been rewritten with the capability to use deep learning and neural networks such as Convolutional Neural Networks. A full review of the autonomous vehicles vision state of the art are gathered in [44] where not only exposes the problems related but the datasets.

2.4.3 Mapping

This module is in charge of creating the best way to represent and the easiest way to analyze the elements detected in the environment in order to process collision-free trajectories. The map displays the real world where the vehicle is moving and is divided into two different types:

- Feature-based map: This type specifies the shape of the environment adding information about corners, walls, rooms and other shapes with the associated locations. For example, if a robot indoor is in an empty room with four corners, a door, and four walls all of them with a specific location, the sensors should detect and acquire easily the exact localization in the digital map. This featured based map is common in indoor robots [18]. In Figure 2.28, the blue lines represent line features, red points represent point features and green arcs represent arc features [53].

- Location based maps: This type of maps represent the real world as a composition of occupied or free space accordingly to the size of detected objects or the sensor

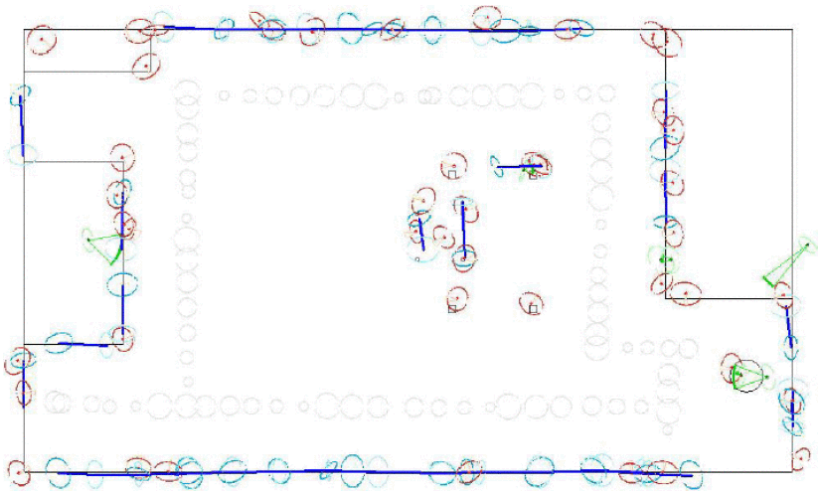


Fig. 2.28 Feature based map where the blue lines represent the line features, red points are features and green arcs are arc features [53].

measurements. One of the most extended representations of the location-based maps is the occupancy grid map where the environment is divided into cells of the same dimension and each cell takes the probability value of being occupied, free or unknown (figure 2.29). For example, in figure 2.30, the lidar sensor returns the readings with high accuracy post filtering and after that, with the distance information for each bean, the corresponding cells take the occupied value and all cells between the closest obstacle and the robot, takes the value-free value. Normally, the standard values are 0 for free cell and 100 for an occupied cell.

It is possible to divide the maps into rotational or static maps. In the first one, the robot is static and the elements move around it. This type of mapping derives into a variety of image transformations of translation and rotation that lose precision in the x-y position of the obstacles and sometimes, the straight lines as walls or buildings are not correctly displayed. The second one is based on the x-y world coordinates that never change as the (0, 0, 0) reference and the robot moves around mapping the environment that will remain in the same position after detecting them.

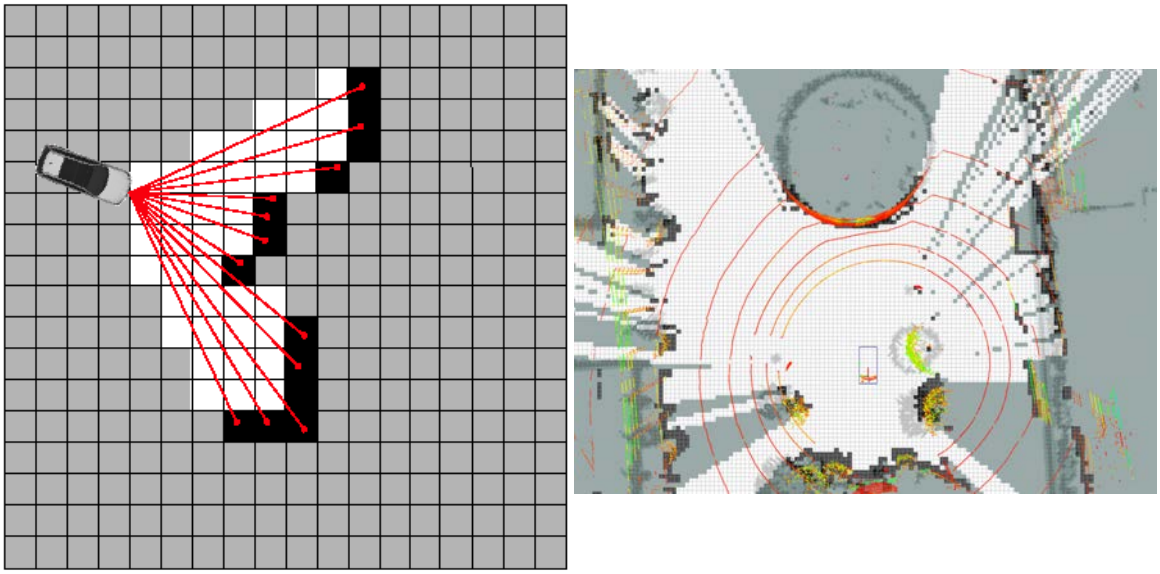


Fig. 2.29 Local map over Global map.

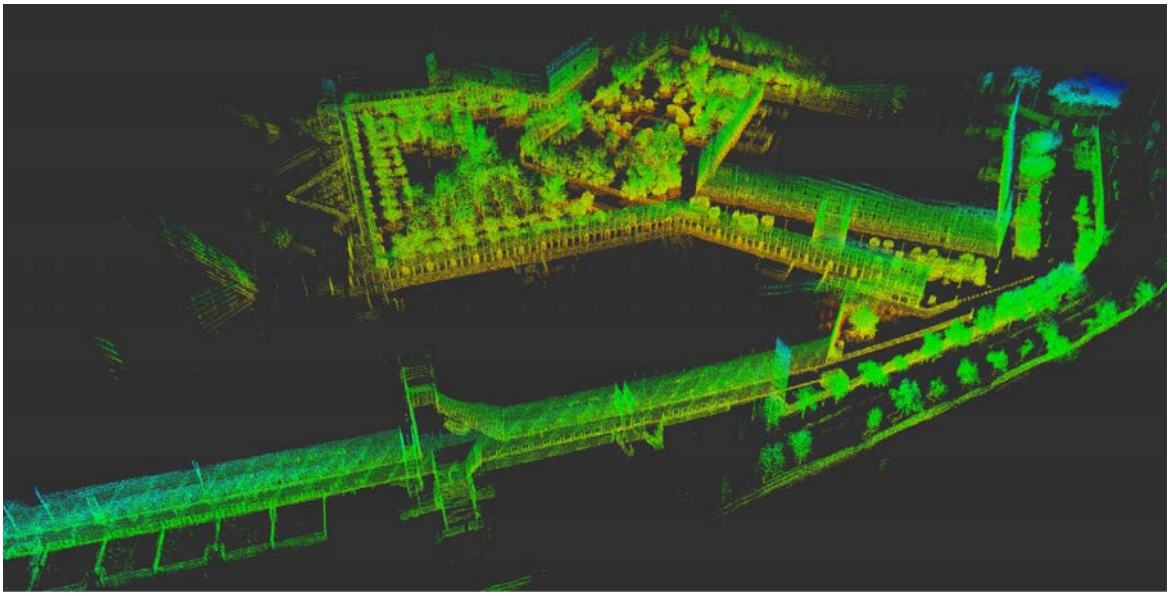


Fig. 2.30 Uc3m 3D map obtained with lidar and odometry.

2.4.4 Planning

Path planning is an optimization problem responsible for finding the optimum path from the starting point and the goal point. It is known as NP-Hardness problem [15] where could be solved by two main procedures: The classical approach which is based on search methods, and Heuristic approach is based on probabilistic algorithms. Between mid 70's to 90's, the classical approaches of search methods were predominant. Different analysis procedures based on graphs were used like Voronoi road maps [2], figure 2.31, Dijkstra algorithm [23], A* and ARA* [31] or potential fields [28], figure 2.32, with the drawback of trapping in local minima and high computational cost requirements [63].

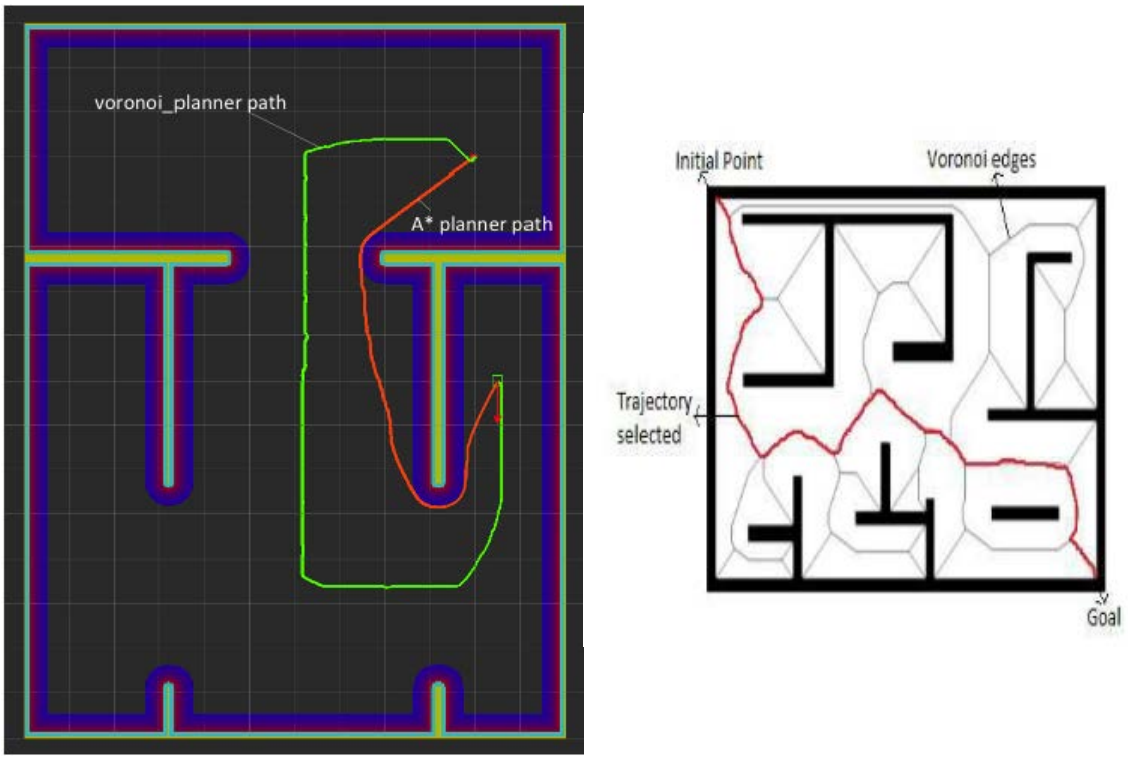


Fig. 2.31 A* vs Voronoi performance.

From posterior years, the heuristic approach becomes more important as a result of the ability to find a solution (not guaranteed) in less time such as Rapidly-exploring Random Tree (RRT) [52], figure 2.33 or Probabilistic Road Maps [46]. Normally, the planning module is divided into global planner, which generates a navigable path obstacle free from the starting point to the goal using a priori knowledge, and local planner which generates a local navigable path taking into account the dynamic obstacles and based on the sensor information. Based on the geometry of the vehicle, there are methods which creates multiple local paths and assign cost values to accept one path and reject the others. A good example is the lattice

planner generation which creates local path using geometric curves such as polynomial curves [19], [22], cubic Bezier [57], splines [91], circular or semicircular forms [84] or clothoids [12] among others. All of them are used in a lattice generation, figure 2.34, giving cost values of success or penalty and select one curve that fits better based on each situation.

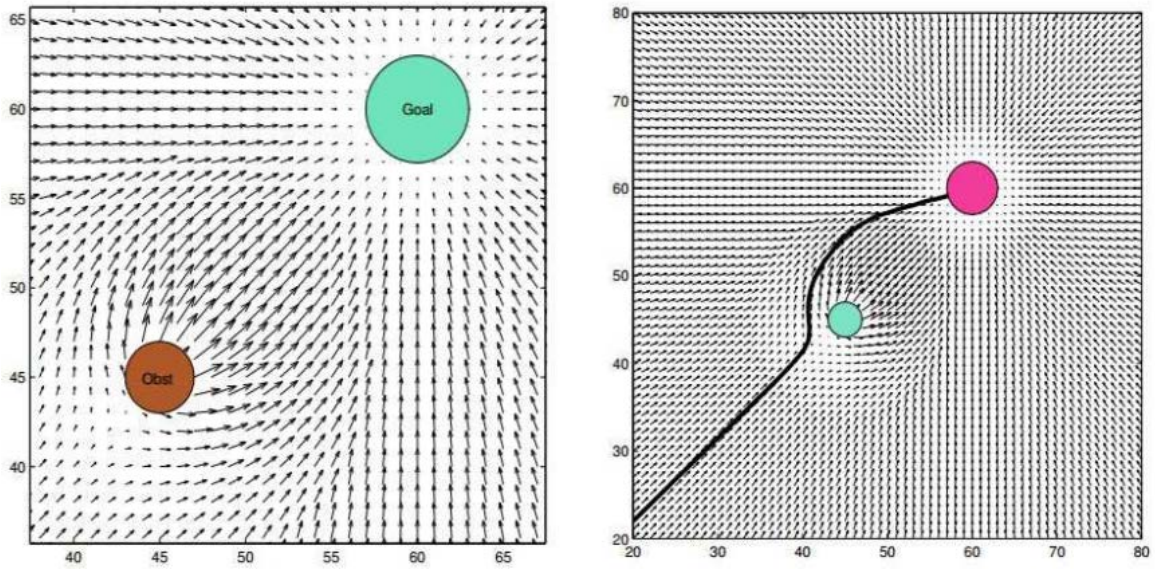


Fig. 2.32 Potential fields obstacle avoidance planning.

Fig. 2.33 Rapidly-Exploring Random Tree with semicircular path planning generation.

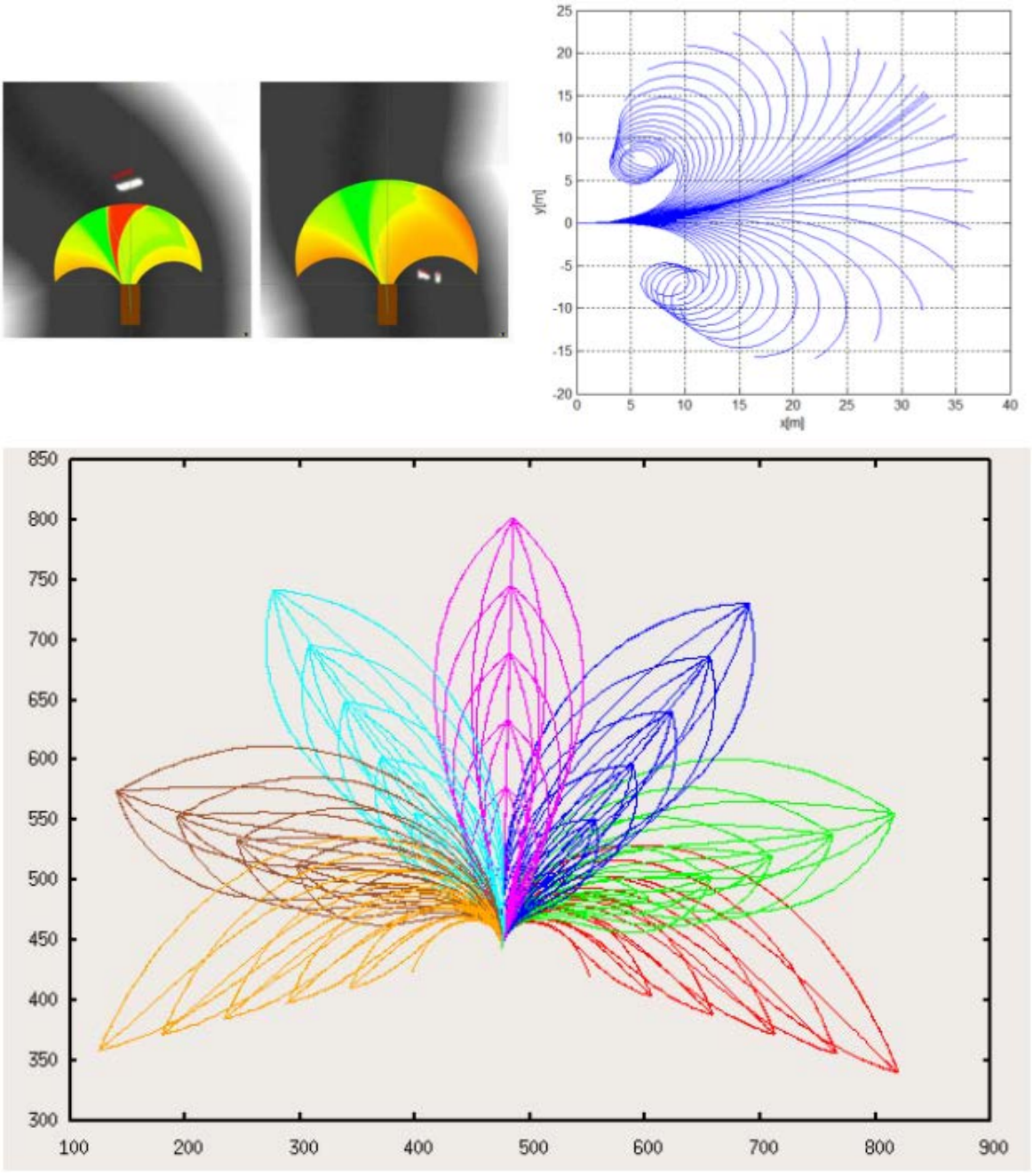


Fig. 2.34 Different lattice planner generation for local planner.

2.5 Examples

The presented examples are very similar in terms of modules due to the fact that all of them need to solve the aforementioned questions such as "Where am I?, Where should I go? and How can I go there?"

2.5.1 Junior 2007

In the Darpa Urban Challenge celebrated in Victorville, California in 2007, the winner was Junior, the vehicle of Stanford University [85]. The architecture of this vehicle was based mainly on localization and obstacle detection due to the urban drive scenarios. In order to navigate, the vehicle had mounted one radar, two IBEO laser-camera, two lidars Sick and one lidar Velodyne for perception and a GPS+compass+IMU for localization in addition of the encoder wheel odometry and velocity. The modules were divided into Sensor interface, Perception, Navigation, User Interface and Global Services. The strongest point of this platform where the sensor fusion for localization and perception. Due to the several lidar sensors based into data Point Clouds, the vehicle was equipped with two powerful computers which processed the data and merge into the static map provided by the competition.

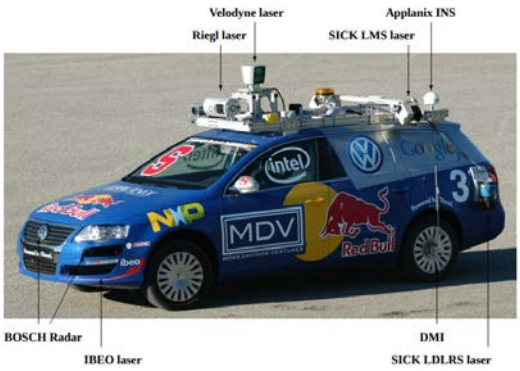


Fig. 2.35 Junior Vehicle.

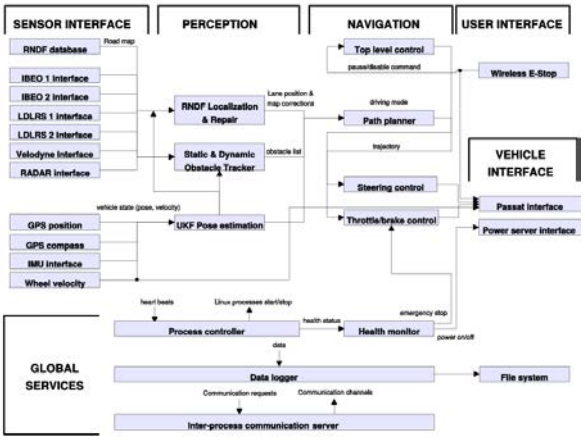


Fig. 2.36 Junior Architecture.

2.5.2 VisLab intercontinental autonomous challenge 2010

This laboratory located in Universita‘ degli Studi di Parma in Italy, reached the goal to drove autonomously 13.000 Km without human intervention from Parma to Shanghai in 2010. The vehicles Piaggio Porter Electric Power 2.37 hold an inertial localization device and GPS, five Lidar sensors and seven cameras to cover the frontal, lateral and rear field of view of the

vehicle [11]. The architecture was simple but effective because was based in the main planner which receives information from the perception layer and trace the best trajectories acting directly in the steering wheel, throttle, and brake to maintain velocity and lateral control. This planner based the decisions on costmaps that classify the information into Waypoints, lanes, obstacles, and ditches.

Fig. 2.37 Junior Vehicle.

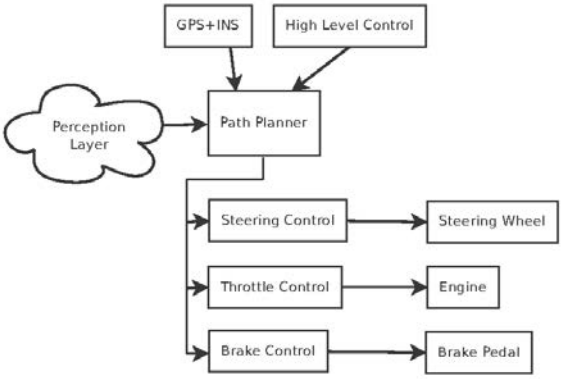


Fig. 2.38 Junior Architecture.

2.5.3 Bertha 2014

This vehicle [98] was the research platform from Mercedes in the year 2014. They mounted GPS, Camera and lidar sensors in order to analyze the surroundings and make an extra effort in the perception module where the cameras took the most important part into the classification and segmentation. The architecture of this platform is divided into Perception, Localization, Planning, Trajectory Control and reactive layer 2.40.

2.5.4 APAChE electric car 2016

Recently, in the Grand Cooperative Driving Challenge celebrated in 2016, the vehicle APAChE from the UTC University [93] used an opensource PACPUS framework that it is available under LGPL-like license. Their general scheme for the architecture involves modules for Control, Localization, and Map, Perception, and Communication, all of them with supervision versus failure. The connection between the sensor and the single powerful computer is done by the CAN and they use HMI to select and monitor the tasks 2.42.

Fig. 2.39 Bertha Vehicle.

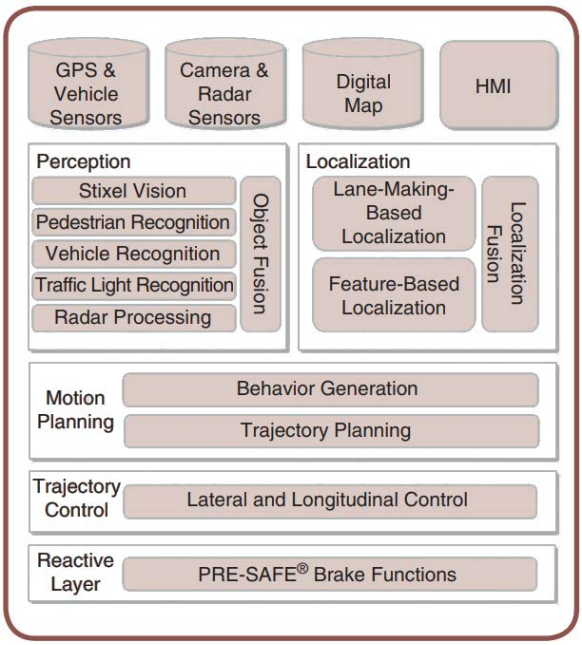


Fig. 2.40 Bertha Architecture.

Fig. 2.41 APAChE vehicle.

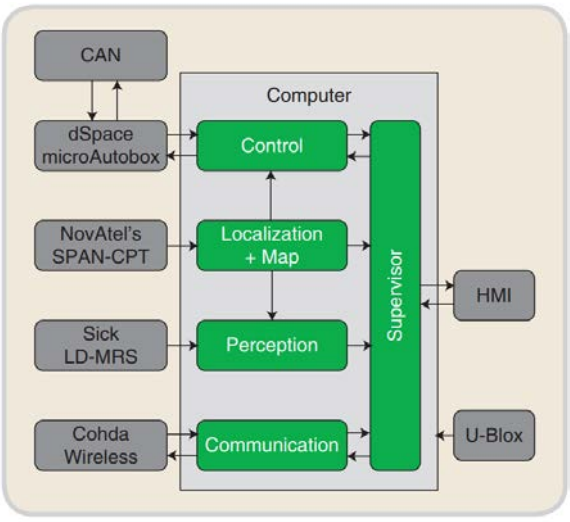


Fig. 2.42 APAChE architecture.

Chapter 3

Hardware Architecture

3.1 Introduction

The iCab platform 1 and 2 showed in figure 3.1 had mounted several sensors in order to obtain the required information of the environment to avoid collisions. Additionally, a screen, router, speakers, light, and buzzer were mounted for debugging and HMI (Human Machine Interaction).

The iCab hardware connections are described in figure 3.2, which is possible to locate in different colors the role of each device. In yellow is described the main actuators for the motors and the encoders such as traction, steering and brake motors and speakers; in red is displayed the sensors connected to each computer; in blue, the network connections over a router; and in green, the input and output for general purposes such as keyboard, mouse and monitor. The arrows indicate the communication between the device and the computer if it is unidirectional or bidirectional and the connection type (PS/2, USB, Firewire, Ethernet, HDMI) for each device.

The use of two computers is necessary to acquire, process and send the navigation commands in a closed loop to calculate the best outcome for navigation procedures. There are unbalanced connections for the sensors because of the high computation load required for lidar and Kinect 2 since both acquire 3D environment information.

The communication between both computers is done by Local Area Network(LAN) using a Router with multiple capabilities as 4G and WiFi in addition to the LAN ports. The configured network is used also to retrieve data from lidar and deliver to the second computer.

Fig. 3.1 iCab research platform 1 and 2.

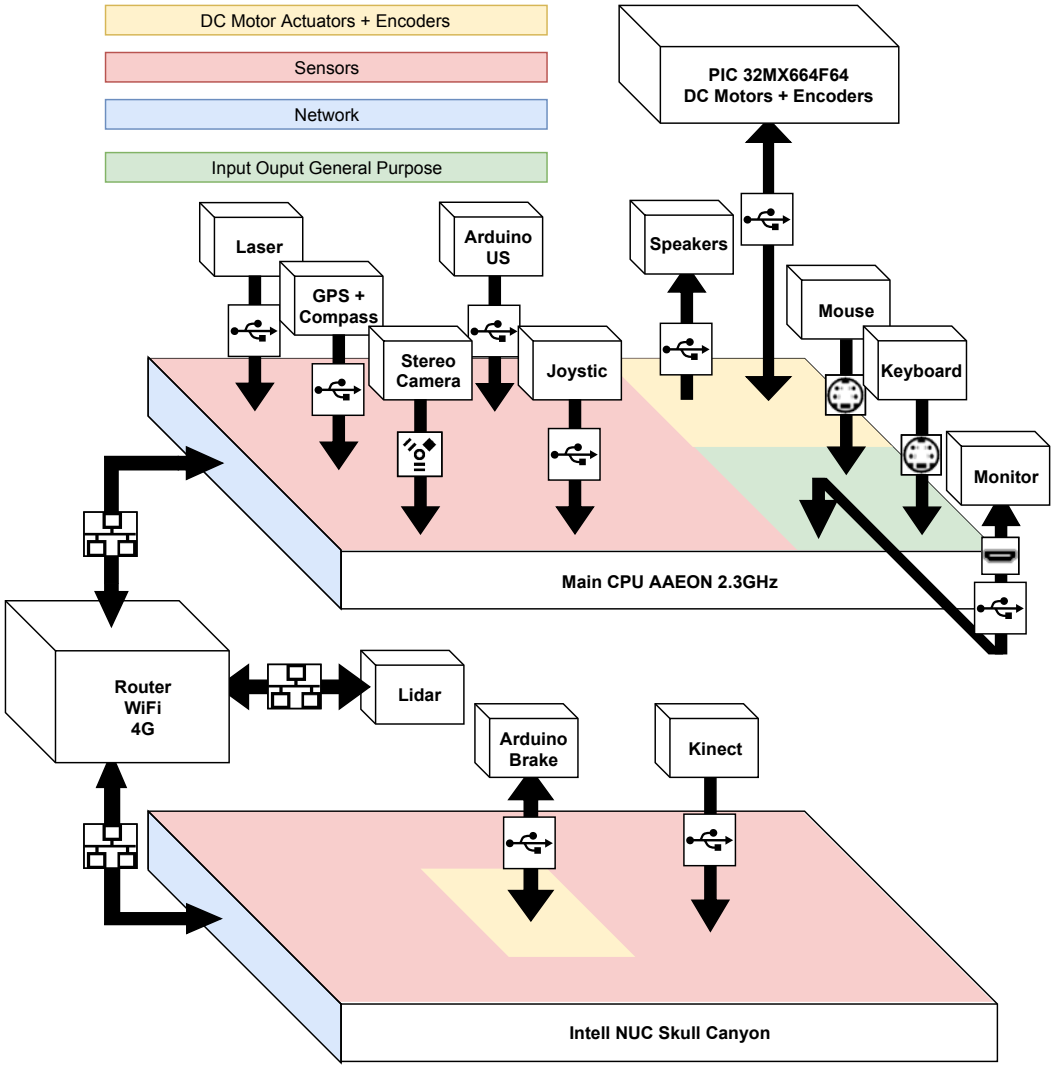


Fig. 3.2 iCab Hardware connection.

3.2 Devices

The stereo camera used in the vehicles are the Bumblebee 2 and xB3 mounted bumblebee2 in iCab1 and xB3 in iCab2. Both are able to generate disparity images using the baseline of 12cm which means in the trinocular camera are only using the left and middle cameras. The position in the vehicle is displayed in appendix A.1 along with the transformation trees.

Fig. 3.3 Bumblebee 2.

Fig. 3.4 Bumblebee xb3.

For the binocular camera is used the image resolution of 640x480 at the frame rate of 10.5Hz after the rectification process. For xB3 the resolution is enhanced to 1280x960 at 15Hz after the rectification process. This device publish the image raw from each camera only and the rectification process is outside of the driver.

The monoplane sensor laser mounted in front of the vehicle is Sick Laser LM293 connected by USB to the main computer. The maximum range in 80m and the configuration is 180 degrees of field of view and each laser beam is separated 1 degree and the refresh rate is 10Hz. The beams measure the distance from the laser to the obstacles.

Fig. 3.5 Sick LM293.

Fig. 3.6 Velodyne.

On the top of the vehicle is mounted the lidar Puck VL16 with 360 degrees of field of view, and deliver the Point Cloud data at 3Hz. In order to create, it has 16 laser layers in +- 15 degrees over the horizontal plane separated 1 degree each layer.

The GPS + IMU + Compass device is managed by Pixhawk and APM 2.6, where the GPS is updating the global position at 2Hz, the compass at 3Hz and the IMU at 50Hz. The GPS device is from 3DR connected to the Pixhawk or APM 2.6 for each vehicle.

Fig. 3.7 Pixhawk.

Fig. 3.8 APM 2.6.

Fig. 3.9 Left image in gray scale.

Fig. 3.10 Left image in RGB color.

Fig. 3.11 Right image in gray scale.

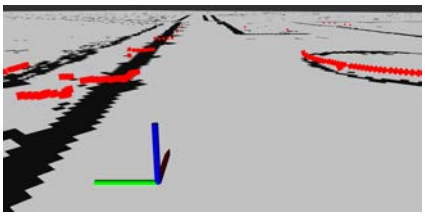


Fig. 3.12 Laser scan.

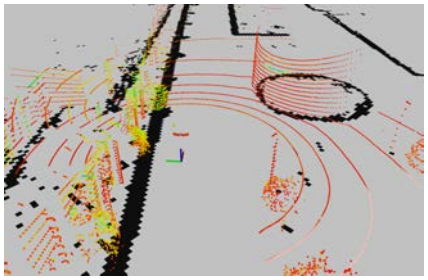


Fig. 3.13 PCL from lidar.

3.3 Low-level control driver

The board designed for traction and direction control uses a microcontroller PIC32MX664F64@80Mhz and serial communication at 1 MHz. The power control for the traction motor uses Mosfet transistors in order to rise and control the amperes required for the motor. Figures 3.14 and 3.15 shows the control board with the microcontroller, serial device, DC/DC converter, and ROM memory to store the configuration and last values.

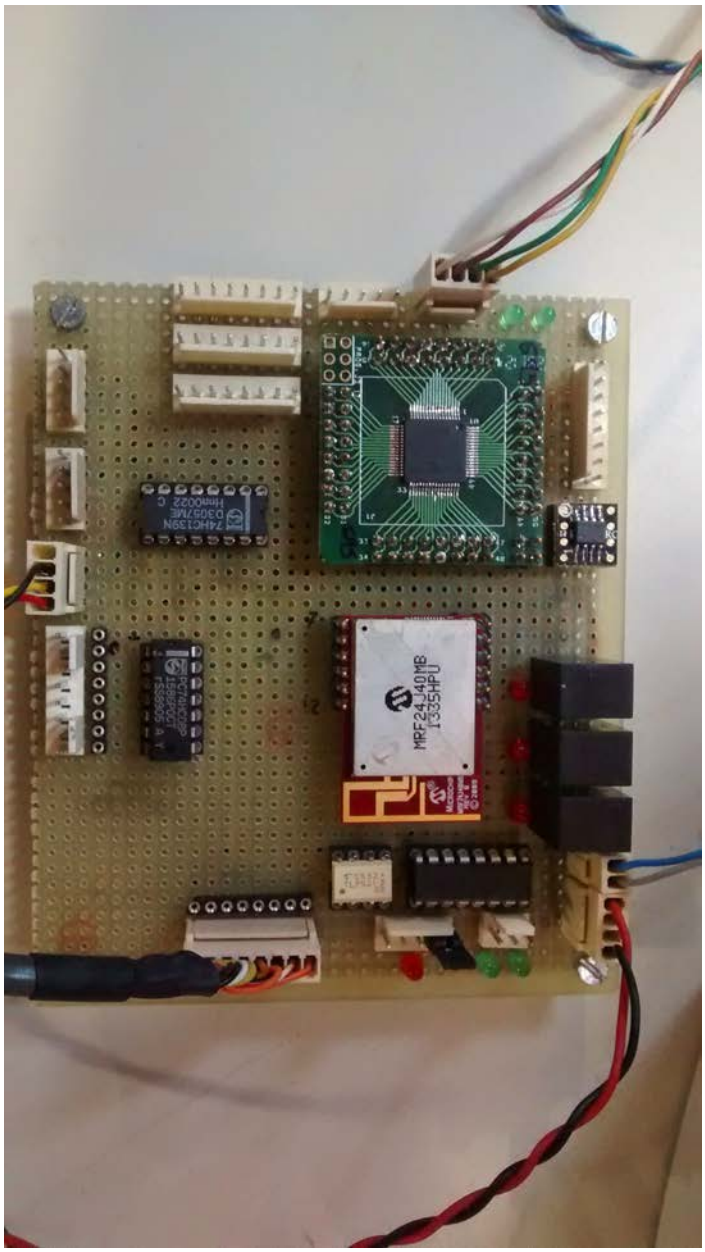


Fig. 3.14 Microcontroller board for managing traction and direction.

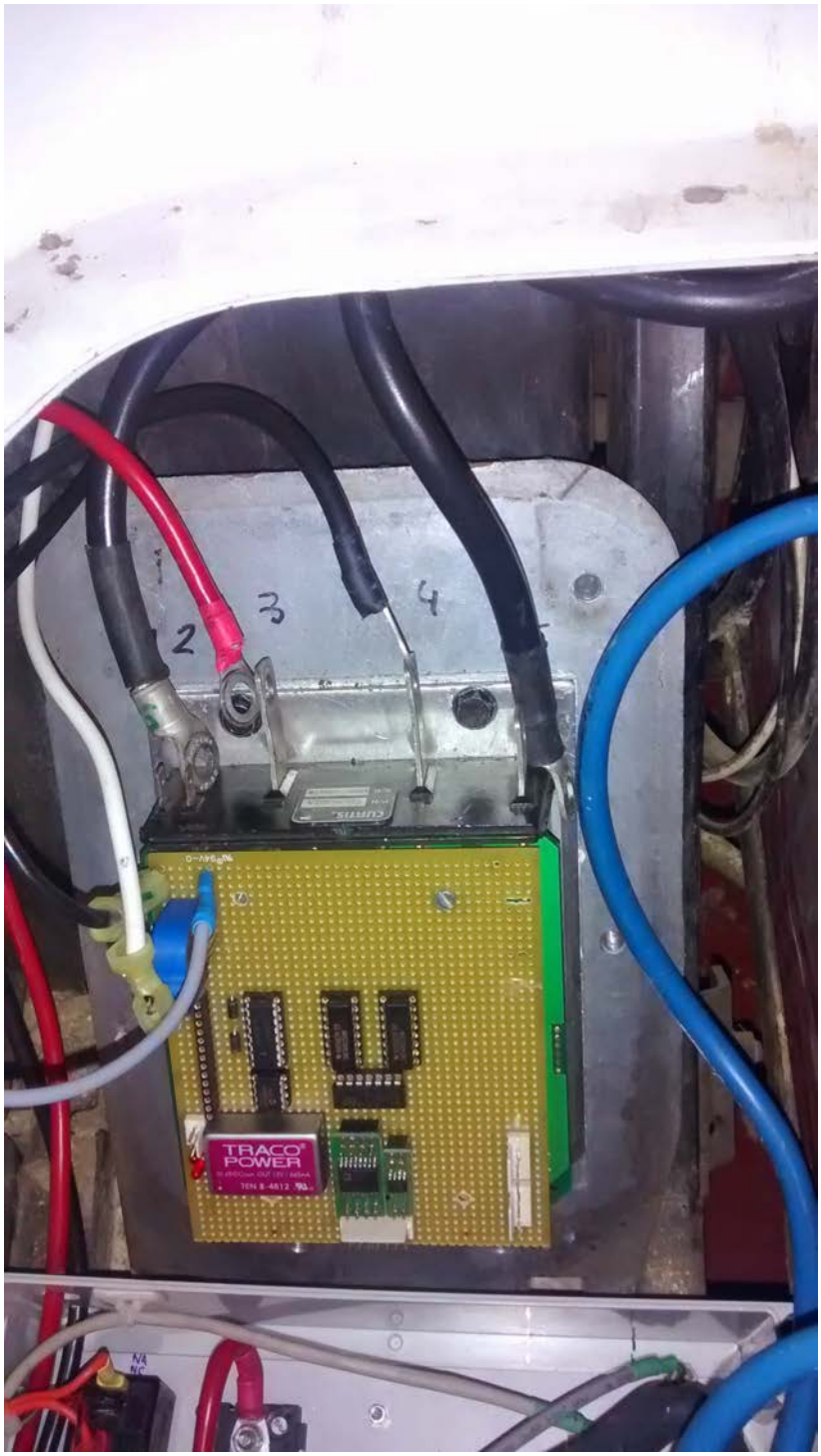


Fig. 3.15 Board for power control.

3.4 Sensor field of view

It is important to acquire information from the environment knowing the limitation of each sensor. In the case of the iCab, the distribution of the sensors and the field of view of each one is described in figure 3.16.

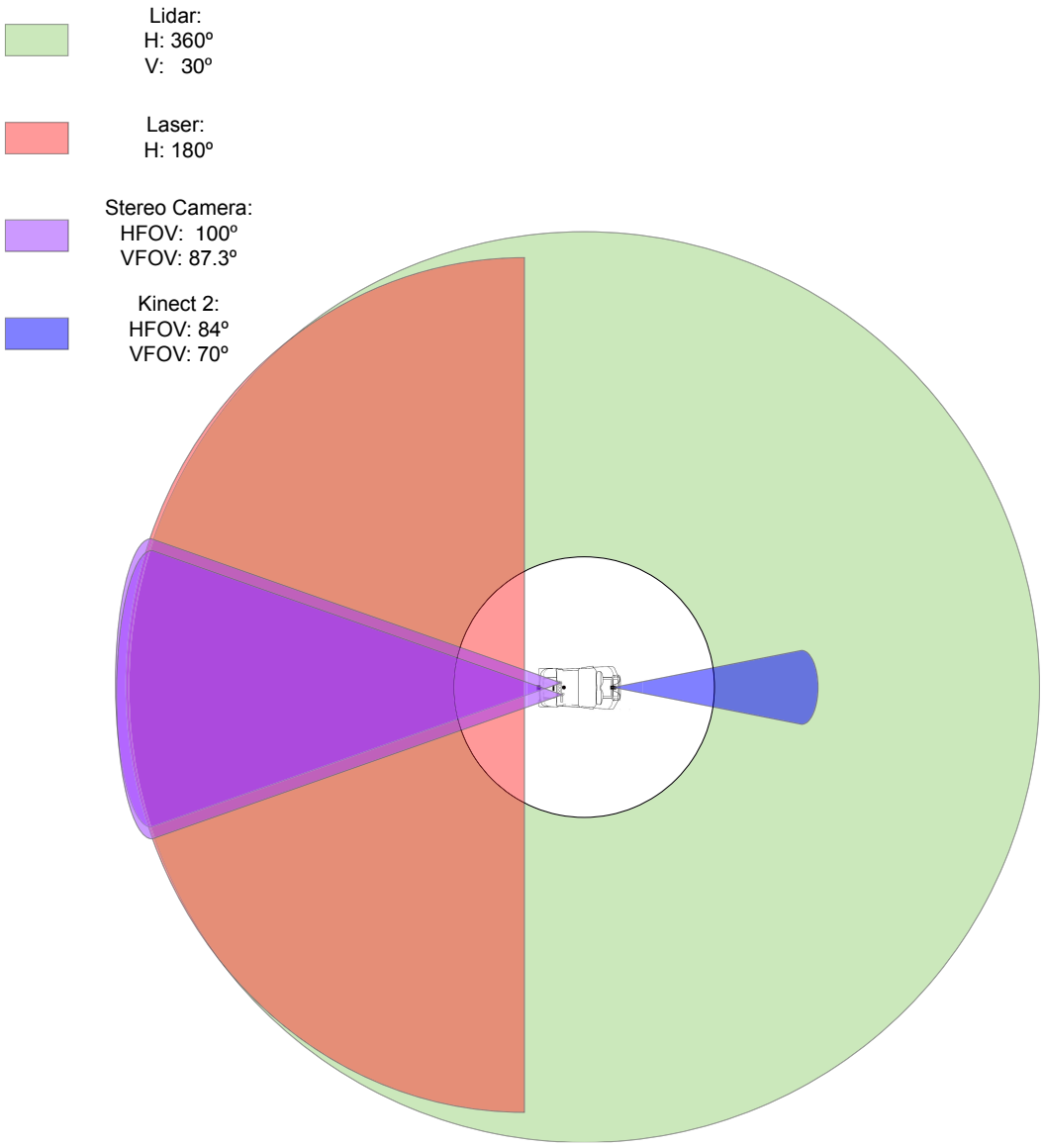


Fig. 3.16 Field of view of the sensors.

3.5 Electrical scheme

The electrical power connections between the devices and the battery are explained in figure 3.17. The battery consists of 6 units of 6 Volts 180Ah serial connected up to 36Volts. In order to connect or disconnect the devices, a switch control has been mounted in the frontal panel of the vehicle with three different configurations. In the first configuration, the DC/DC converter of CPUs and Sensors are activated, in the second configuration, the CPUs DC/DC converter is activated alone, and in the third configuration, both converters are disabled. The circuit power supply is controlled by the Key of the golf cart provided by the manufacturer and it allows or cuts the energy provided to the traction and direction motors. Notice that the stereo camera and GPS+IMU+Compass are directly connected to the AAEON CPU by USB or firewire and do not require an additional power supply. Additional pictures of the electric connections mounted on the vehicle are shown in figures 3.18, 3.19 and 3.20.

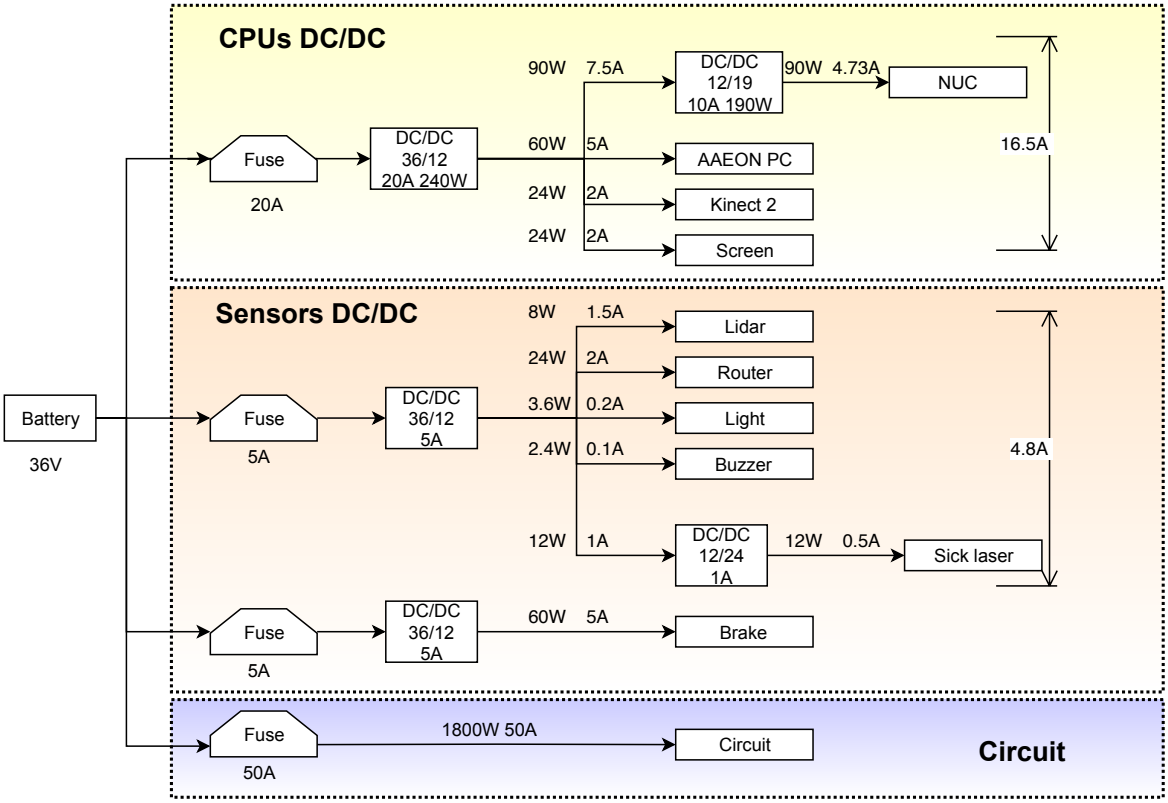


Fig. 3.17 iCab power scheme.

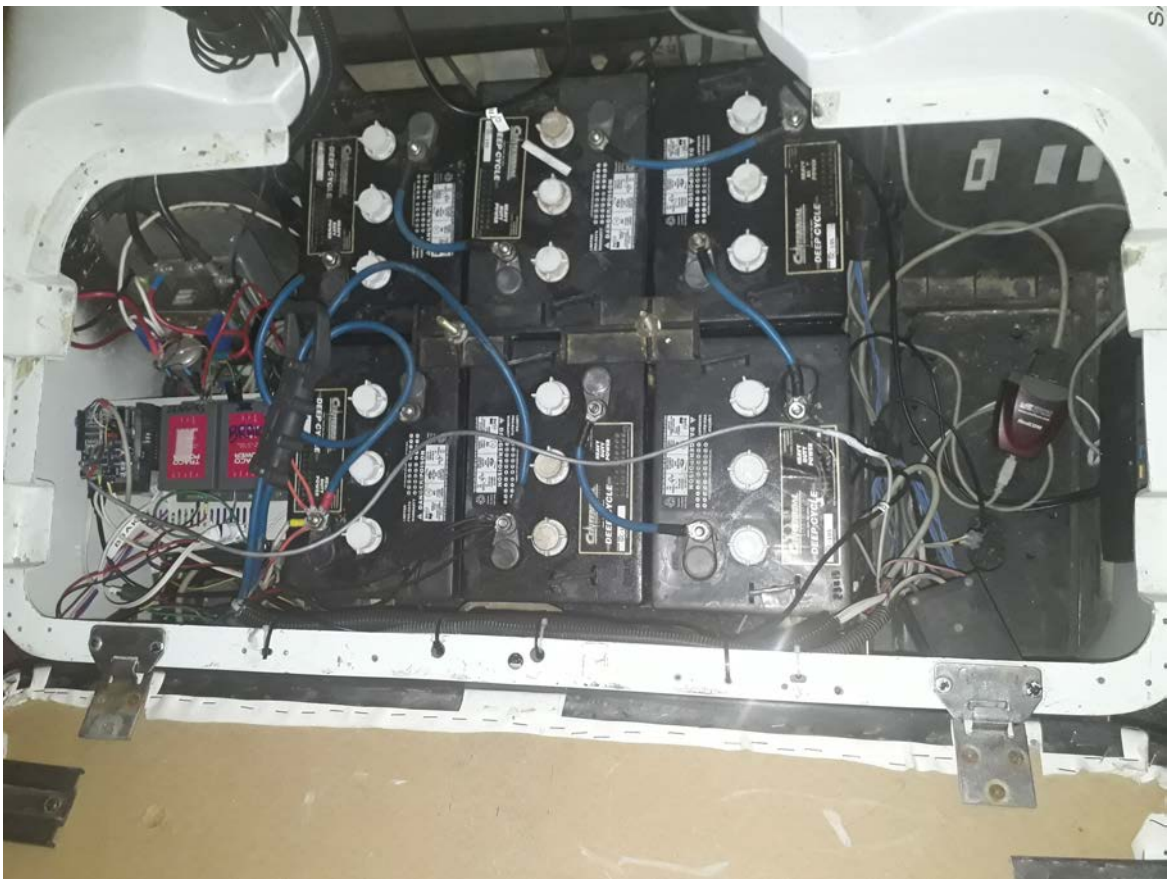


Fig. 3.18 iCab batteries NUC and Arduino brake with DC/DC converters and circuit.

Fig. 3.19 NUC auxiliary CPU.

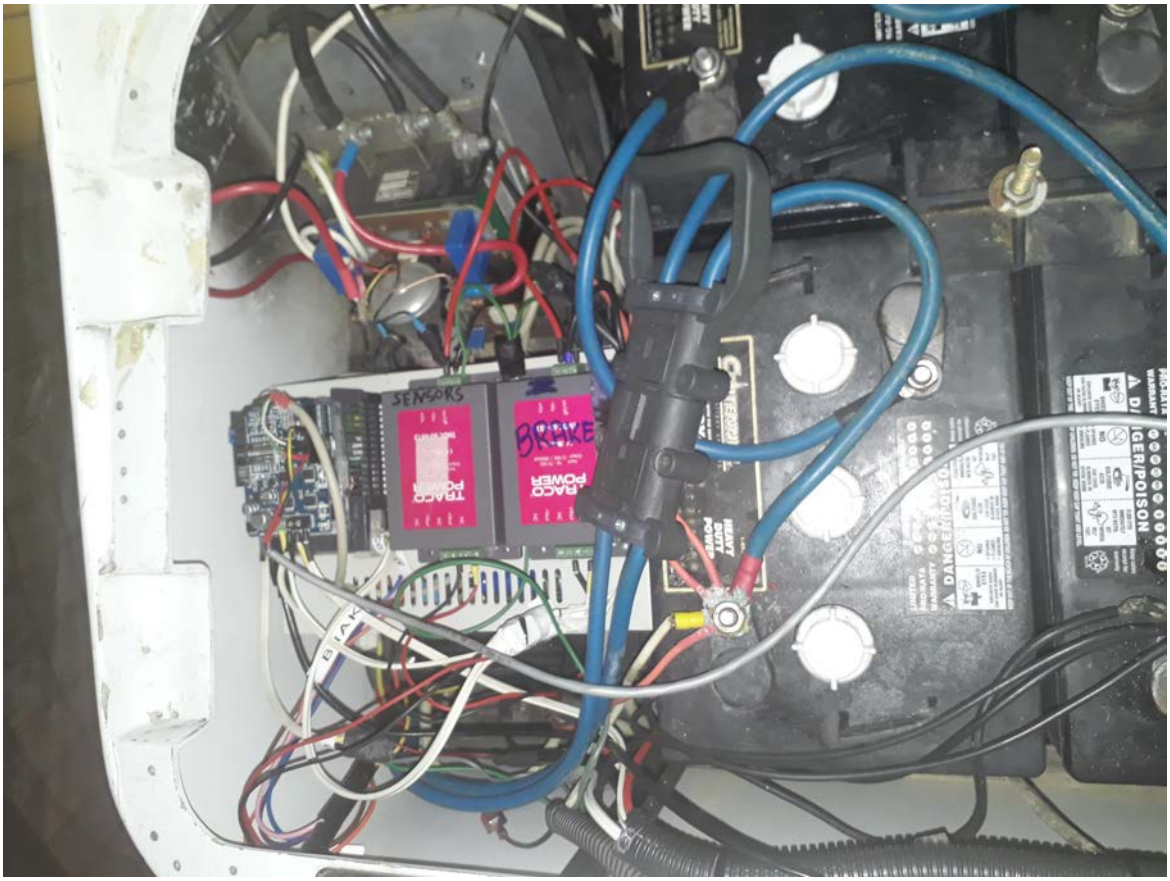


Fig. 3.20 Low level circuit for traction and direction, Arduino brake system and DC/DC converters.

Chapter 4

Navigation modules

This section is intended to explain the basic modules involved in the movement and navigation of the vehicle. There are multiple disciplines involved in autonomous vehicles developing and most of them are related to the ability to safely move from one point to another. Four essential problems are solved using control, localization, perception, and path planning. Additionally, Human Machine Interaction such as GUI or sound manager is included as extra modules to complete the architecture.

4.1 Control

The low-level control is attached to the platform where the architecture is running and is specific for each vehicle. The movement is done by two motors, one for the longitudinal displacement and the other one for the steering wheel. Almost all vehicles are based on Ackermann model [36] and in order to simplify the control module, in this project has been used the simplified Ackermann model as shown in the figure 4.1. Both direction wheels angle is denoted by δ_i for inner wheel and δ_o for the outer wheel of the arc. The w denotes the distance between wheels in the same axis and l denotes the distance between the axes of the frontal and rear wheels. This distance l is divided into l_f and l_r in order to take into account the center of mass of the vehicle. R is the total radius of the arc from the center of mass of the vehicle.

Fig. 4.1 Ackermann model scheme.

The equations related to the movement of the vehicle 4.1 associates the steering angle δ for the simplified Ackermann model with the desired arc with radius R.

Ackermann's equations:

$$\delta_o = tan^{-1} \frac{l}{(R + \frac{w}{2})}, \tag{4.1a}$$

$$\delta_i = tan^{-1} \frac{l}{(R - \frac{w}{2})}, \tag{4.1b}$$

$$\delta = \frac{\delta_o + \delta_i}{2} \cong \frac{L}{R} \tag{4.1c}$$

(4.1d)

Steering wheel

The control for the direction of the vehicle is done by a motor-encoder system as shown in the figure 4.2, where the steering wheel has been removed and replaced. This motor Parvalux PM60 LWS brushed DC incorporates a reduction gears that increase the torque and reduce the speed. The platform that holds the motor is linked to the chassis and the output is directly connected to the rod for the rack and pinion. In order to measure the movement of this motor, it has been selected an absolute encoder TEKEL TKM60, with 8192 positions per 4096 turns. The transmission of movement to the encoder is done by a timing belt with the reduction ratio of 36/60 for the gears in order to avoid the overflow between top angles of the encoder. Finally, for the electrical signals, an H-Bridge of MOSFET transistors to allow high power supply has been implemented as a driver. So, a microcontroller PIC32MX664F64@80Mhz with analog ports for measuring the instantaneous current along of digital ports for the encoder govern all this system, whose its high-frequency operation cycle, allows low-level control at 100Hz.

Fig. 4.2 Substitution of the steering wheel for motor-encoder system.

There are two types of steering control for the motor: in position and in velocity using the encoder sensor. For the position control (see figure 4.3), an extra performance is added in order to save energy and block the power consumption when the angle reaches ± 0.3 degrees of the reference value, that means there is an intended steady state error. For the

velocity control of the steering angle, the reference value is provided into degrees per second which are compared to the current velocity and the generation of the PWM is done by a PID controller (see figure 4.4).

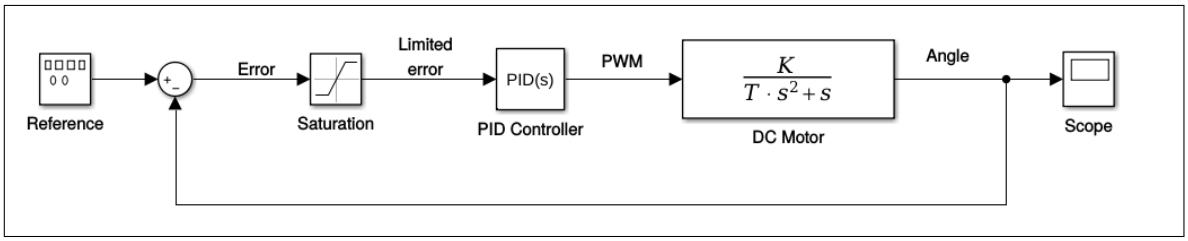


Fig. 4.3 Angle control loop for steering angle.

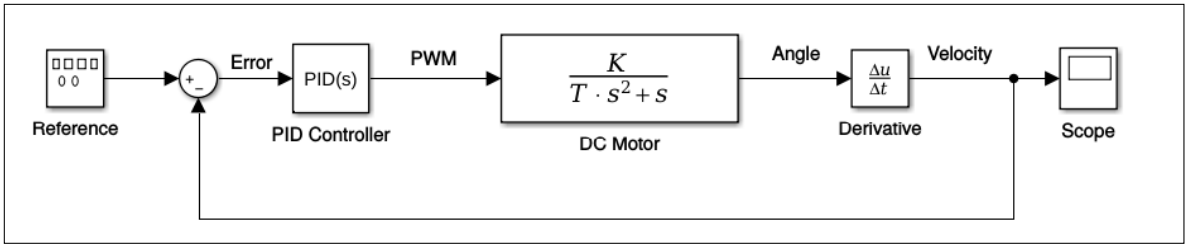


Fig. 4.4 Velocity control loop for steering angle.

This steering control is done in the microcontroller outside of the main ROS framework using a serial port interface by USB wire to send the custom messages for the type of control and reference.

In order to explain the behavior of steering angle in the iCab, it has been moved the directional wheels from 0 degrees to a different value specified in the table 4.1 testing small and big steps. The results are displayed in figure 4.5, where the current angle is displayed in blue, and the reference angle is displayed in red. For better understanding, the zoom in is displayed in figure 4.6.

Another example of the steering angle control is displayed in figure 4.7 where the system is configured on autonomous mode. The control commands are shown in blue.

Table 4.1 Set of steering angle values to analyze the behavior for the steering wheel

Reference (degrees)	Final value (degrees)	Steady state error (degrees)	Overshoot (%)
0.0	0.3	0.3	-
5.0	4.7	0.3	-
5.1	4.9	0.2	-
5.2	5.0	0.2	-
5.3	5.0	0.3	-
5.2	5.0	0.2	-
5.1	5.0	0.1	-
5.0	5.0	0.0	-
-5.0	-5.1	0.1	-5.6 (6%)
-5.1	-5.1	0.0	-
-5.2	-5.1	0.1	-
-5.3	-5.1	0.2	-
-5.2	-5.1	0.1	-
-5.1	-5.1	0.0	-
-5.0	-5.0	0.0	-
10.0	10.1	0.1	10.5 (3.3%)
10.1	10.1	0.0	-
10.2	10.1	0.1	-
10.3	10.1	0.2	-
10.2	10.1	0.1	-
10.1	10.1	0.0	-
10.0	10.1	0.1	-
5.0	4.8	0.2	4.7 (6%)
0.0	-0.1	0.1	-0.5 (10%)

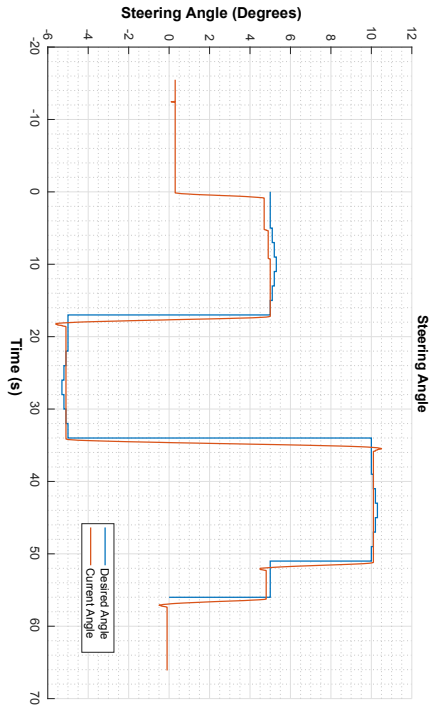


Fig. 4.5 Current steering angle in blue; Desired angle in orange.

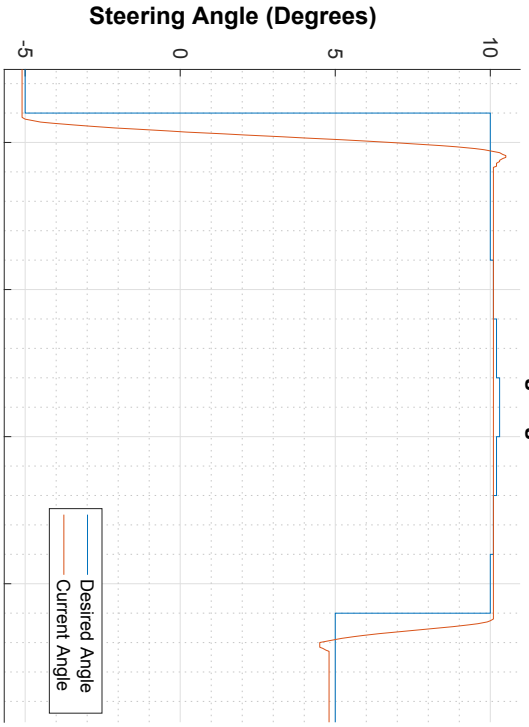


Fig. 4.6 Zoom to appreciate in detail the steady state error and the overshoot.

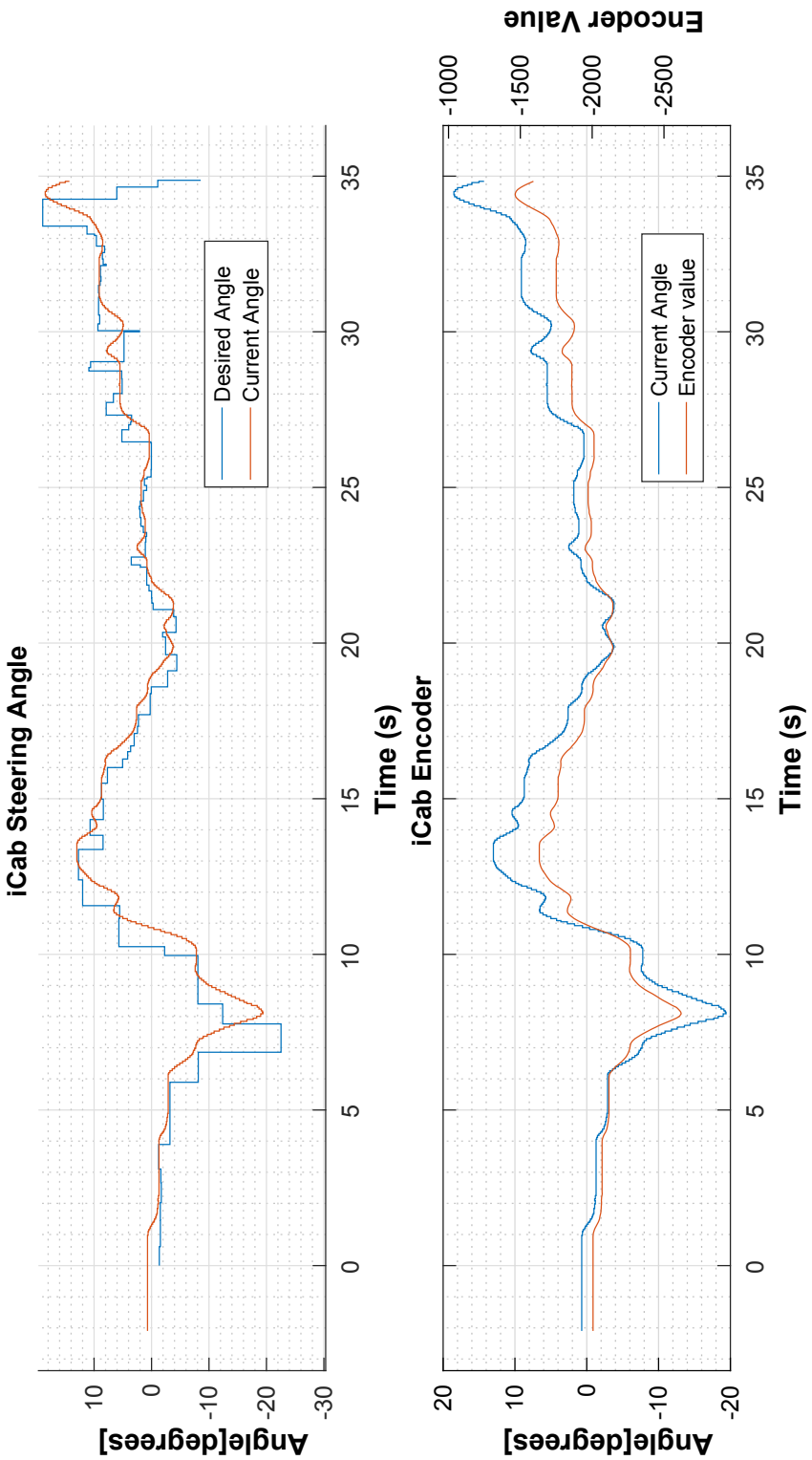


Fig. 4.7 Performance in an autonomous mode experiment.

Traction

The control of the linear velocity is done by a PID controller, figure 4.8 which takes the velocity reference as input and compare it with the current velocity. The error (e_{vel}) is used to generate a PWM (equation 4.2) using proportional P gain, derivative D gain and Integral I gain as follows:

$$pwm = P_{gain} \cdot e_{vel}(t) + D_{gain} \cdot \frac{\delta(e_{vel}(t))}{dt} + I_{gain} \cdot \int e_{vel}(t) \cdot dt \tag{4.2}$$

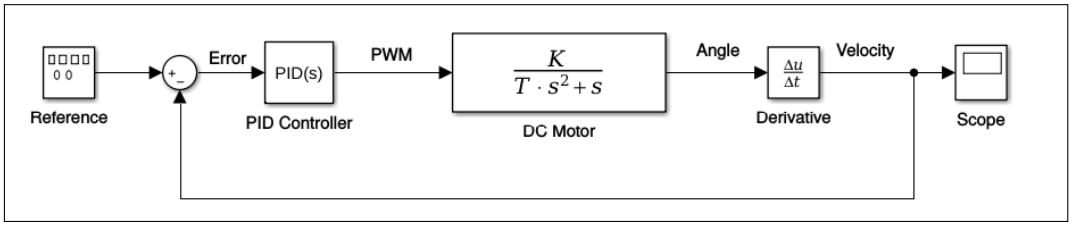


Fig. 4.8 Velocity control loop for steering angle.

The current velocity is obtained using an incremental encoder. The ticks value (n) of the encoder is processed using the time lapse between readings (δt) and is directly related to the radius of the wheel, see equation 4.3.

Linear velocity:

$$mpt = \frac{2\pi R}{C_e}[m/tick], \tag{4.3a}$$

$$distance = n \cdot mpt[m], \tag{4.3b}$$

$$velocity = \frac{distance}{\delta t}[m/s], \tag{4.3c}$$

Where:
mpt = meters per tick,
R = wheel radius,
C_e = ticks per revolution (encoder resolution),
n = number of ticks.

In the figure 4.9, the desired speed, in blue, is the velocity reference. This PID controller is generating the PWM value in order to try to follow the reference. figure 4.10 shows the linear acceleration measured by the IMU in blue and filtered in red.

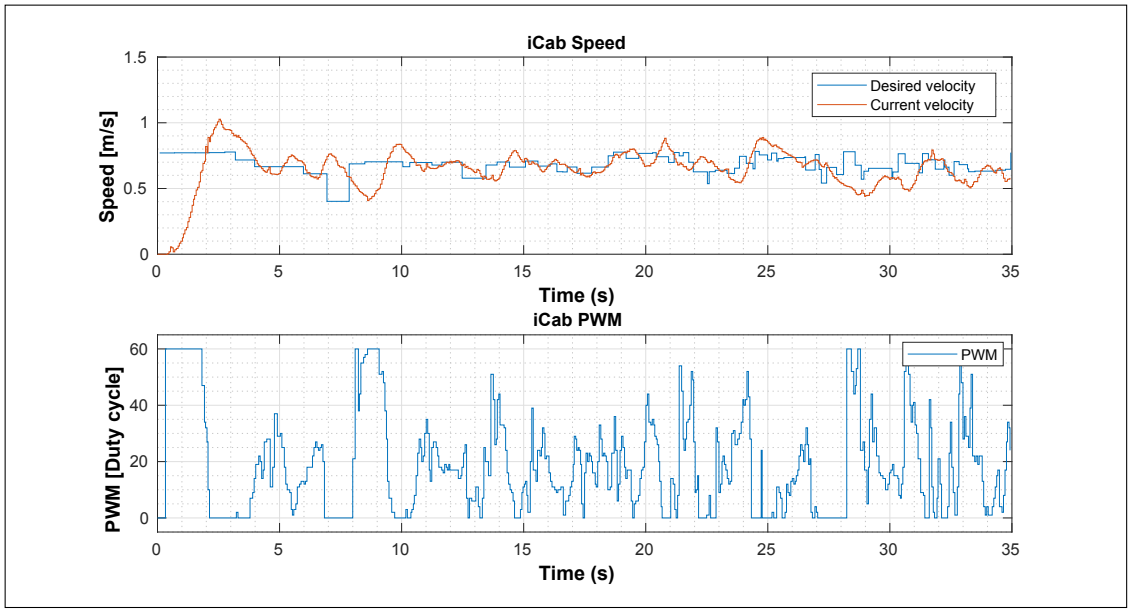


Fig. 4.9 Desired velocity in orange; Current velocity in blue.

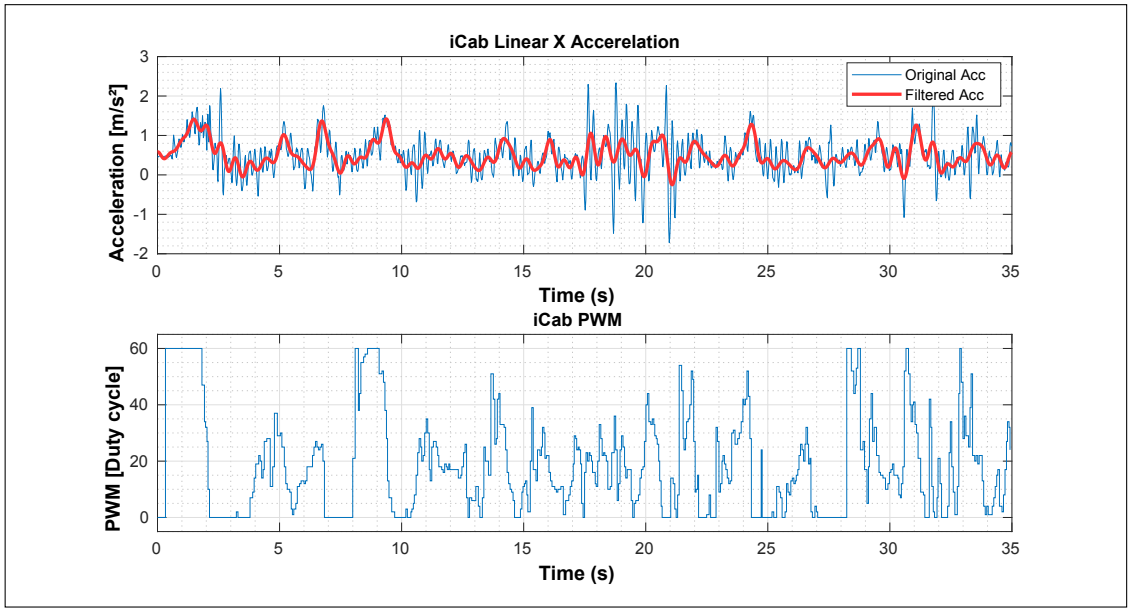


Fig. 4.10 Acceleration.

4.1.1 Traction brake

The linear control is complemented by the brake located under the vehicle which activates the brake pedal, figure 4.11. It is configured to brake in case of emergency when the user presses the panic button or there is error communication between the driver microcontroller and the main PC.

Fig. 4.11 Traction brake device.

4.2 Localization

The most critical module in navigation is the digital knowledge of the position of the vehicle in the environment. The real movement of the vehicle is measured by different algorithms in order to compare and fuse. All these modules are implemented in the architecture and running in parallel. Each method has been designed to create individually the odometry with comparison purposes and later using filtering and fusion, improve the estimated local odometry.

During this module is used one scenario to expose and compare the odometry results obtained with different methods and sensors. This scenario is the Sabatini building(see figure 4.12) where the vehicle performs a U-shape trajectory. This trajectory is repeated four times to finally end at the same starting position.

There is two types of localization (see figure 4.13):

- Local: which uses as reference the initial point where the vehicle is located as $[local_X, local_Y, local_\theta] = [0,0,0]$.

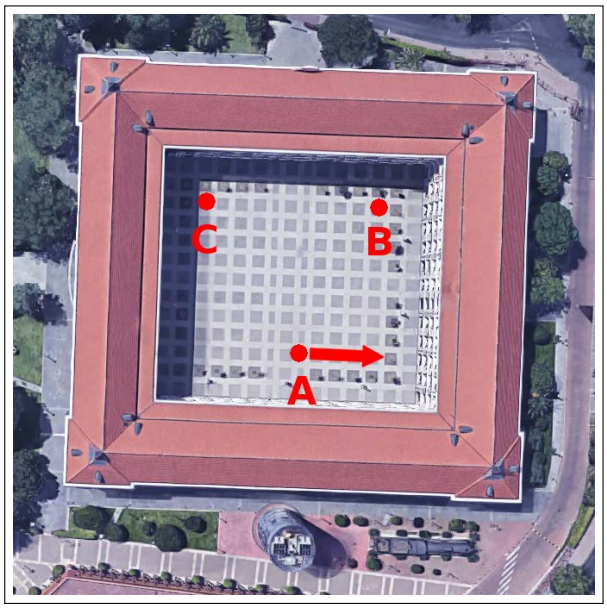


Fig. 4.12 The trajectory starts in A point and the vehicle is heading in the arrow direction. The vehicle advances and performs a left turn until it reaches the point B. When the point B is reached, the vehicle turns 180 degrees using the left. After that, it performs two right turns in each corner, passing through the point A until it reaches the point C, where it performs another 180 degrees turn using the right. Finally, when it reaches the corner again, it turns to the left and advance to the point A. Without stopping, this full cycle is repeated four times.

- Global: which uses as reference a fixed world point previously defined. This fixed point is related with the global map used in the mapping subsection. $[global_X, global_Y, global_\theta] = [0,0,0]$.

In order to determine the global localization between the vehicle and the fixed world point, exists the initial localization. This initial localization is the distance between the global fixed coordinates (\map) and local fixed coordinates (\odom) and it is useful in order to determine the global odometry when the vehicle moves around the world.

Encoder odometry

It is the simplest local odometry technique but the least reliable of all of them due to the accumulate error providing by the lack of accuracy of values such as wheel circumference, encoder ticks per revolution and slippery. These errors affect the linear distance covered by the vehicle and it is highly sensitive to accumulate heading errors. A slight deviation from the real heading angle resulted in really poor performance and also these errors are really difficult to estimate due to the different roads, pavements, and materials.

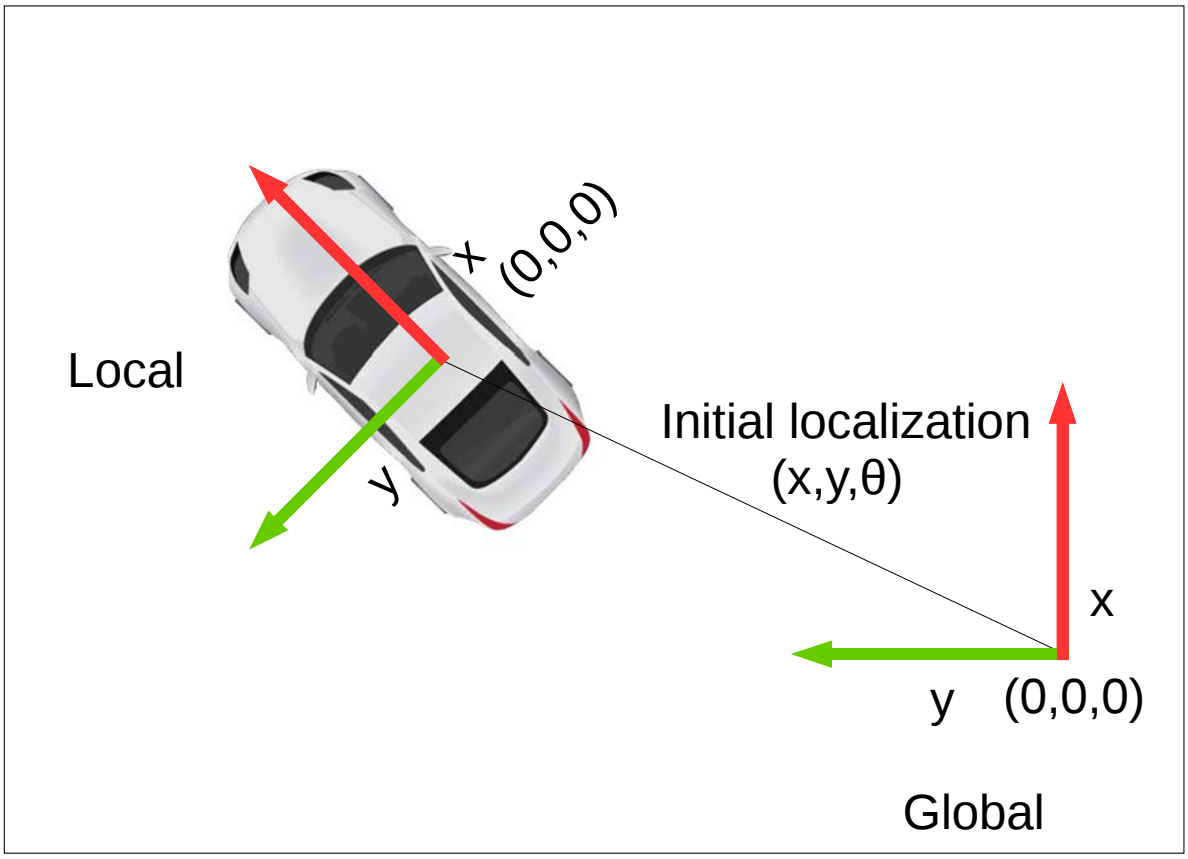


Fig. 4.13 Global and local reference points including the initial localization.

The algorithm used in this module generates the local X, Y, θ movement, from the starting point (0, 0, 0) in the local coordinates using rear wheel and steering wheel encoders and the compass. The initialization step for this method stores the initial values of rear wheel encoder ticks, steering angle and compass angle. In the main loop, the covered distance is obtained with equation 4.4b. For each cycle, the algorithm uses the variations (δ) of the movement and accumulate them to compose the odometry (equations 4.4e, 4.4f and 4.4e)

In order to use the Kalman Filter (Extended or Unscented), it is mandatory to provide the covariances in the three dimensions. This task is not easy because working with real vehicles means real measurements over at least two dimensions flat world (X, Y). These real and exact measurements are known as ground truth and for this project, we don't have one. Instead of using the real covariances for encoder odometry, this module uses the online

adaptative covariance estimation [72] which generates the estimate noise covariance using the known uncertainty of GPS sensor.

Local odometry using wheel encoders:

$$\Delta_{ticks}(t) = ticks(t) - ticks(t-1) \tag{4.4a}$$

$$\Delta_{distance}(t) = \Delta_{ticks}(t)\frac{2\pi R}{C_e} \tag{4.4b}$$

$$\Delta\theta(t) = \frac{\Delta_{distance}(t)}{L}\tan(steering\_angle(t) \tag{4.4c}$$

$$header\_angle(t) = compass(t) - compass(0) \tag{4.4d}$$

$$local_X(t) = local_X(t-1) + \Delta_{distance}(t)\cos(local_\theta(t) - \frac{\Delta\theta(t)}{2}) \tag{4.4e}$$

$$local_Y(t) = local_Y(t-1) + \Delta_{distance}(t)\sin(local_\theta(t) - \frac{\Delta\theta(t)}{2}) \tag{4.4f}$$

$$local_\theta(t) = local_\theta(t) - \Delta\theta \tag{4.4g}$$

The results using the encoder odometry is compared at the end of this subsection in order to appreciate better the outcome of all odometries. Figure 4.14 displays an example of the results obtained with the encoder odometry. It is possible to appreciate the path followed by the vehicle.

Visual odometry

This odometry is based on the image processing. As said in the previous chapter, is possible to extract movement information analyzing frame by frame the position of elements in the image. This analysis needs pre-processing the image to clean, filter, extract features and after that, search this features and calculate the movement in the consecutive frames. Due to the fact that this method is based on accumulative movement, as happens in the encoder odometry, a minimal deviation in translation or rotation produces a cumulative error. For this project, the visual odometry method is part of the work [67] with the improvement of [66] and is based on the analysis of UV-disparity map, extraction of features and calculating the movement of the vehicle.

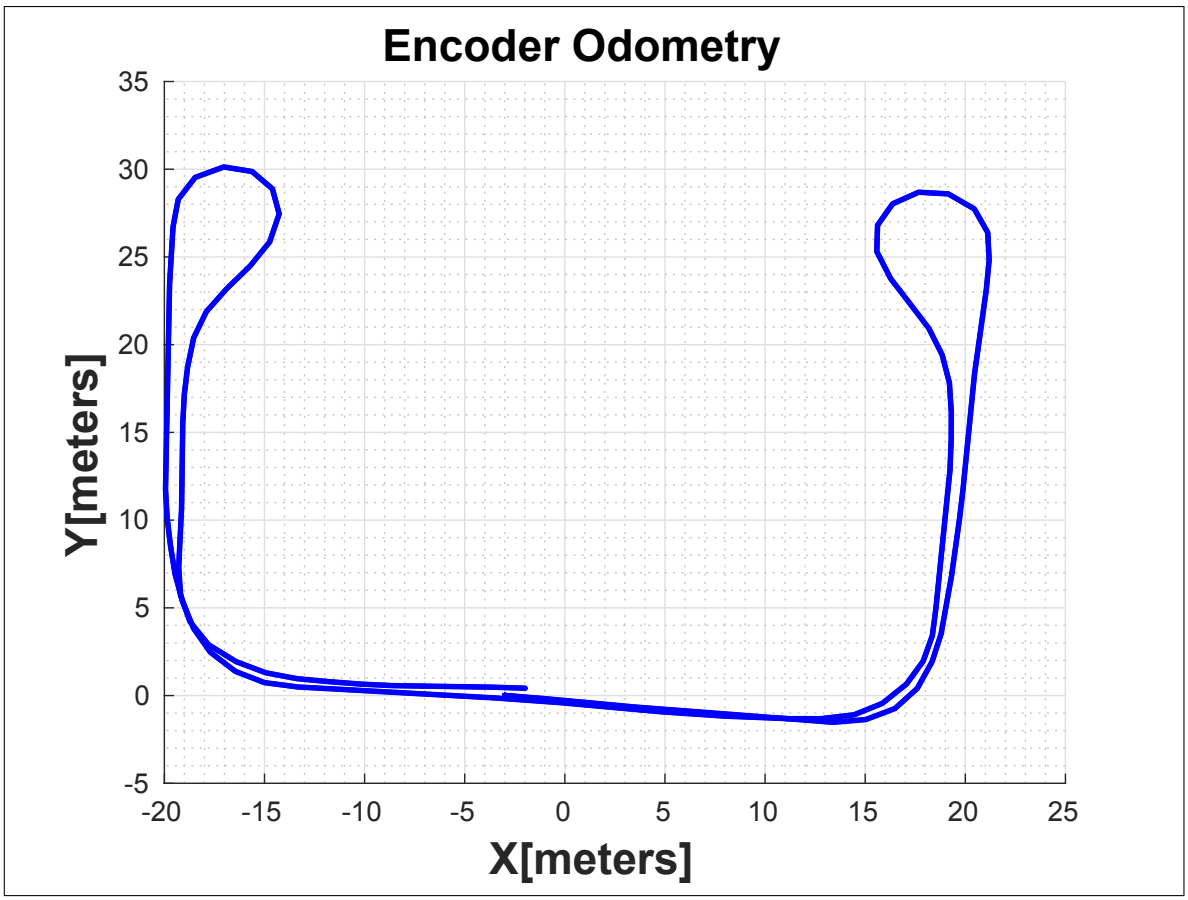


Fig. 4.14 Encoder Odometry outcome

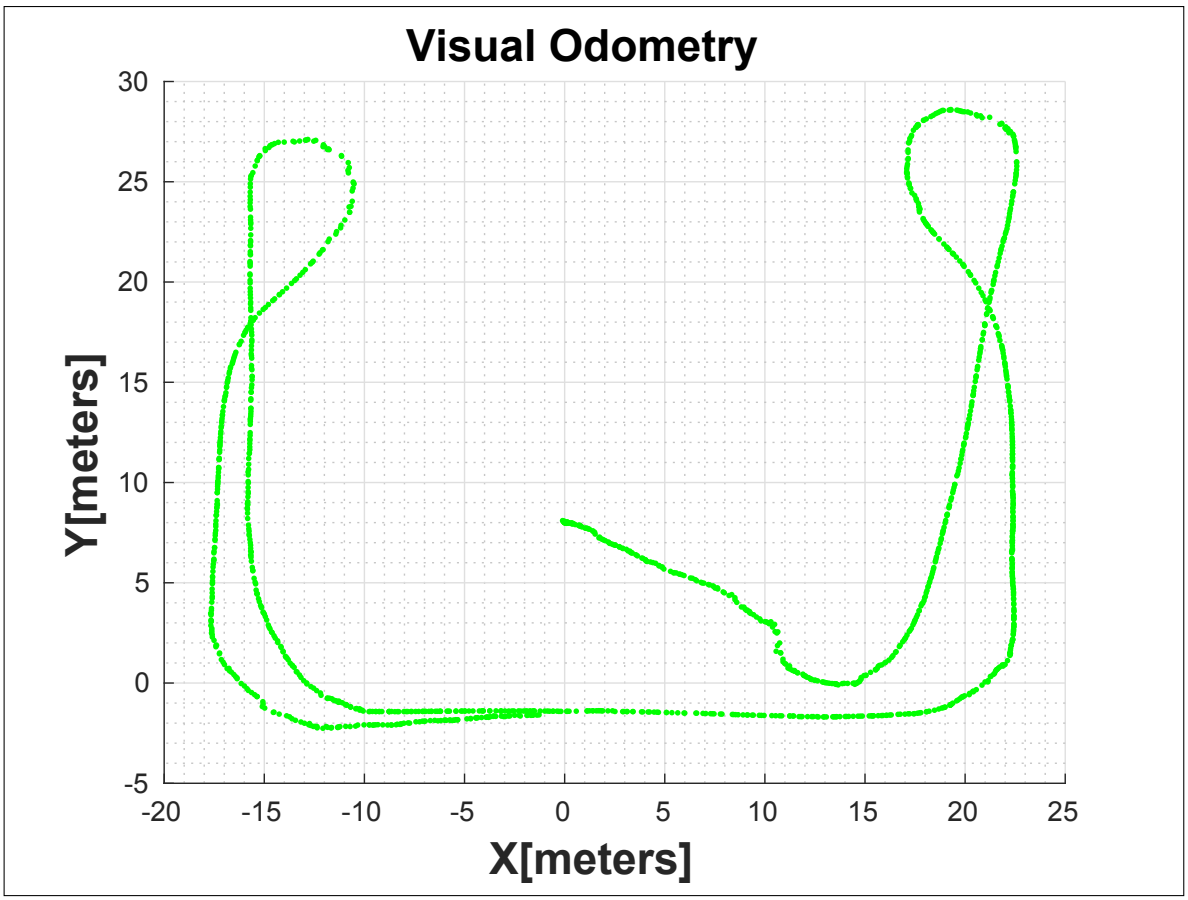


Fig. 4.15 Visual Odometry outcome

Laser - Lidar odometry

This module process the lidar information 3D to generate the digital 3D movement of the vehicle. The algorithm used is part of the work [96] where features as edge points and planar points are tracked from one scan to the next. For each iteration, the algorithm seeks the correspondence of this features in both scans until the nonlinear optimization converges. This method is normally ICP (Iteration Closest Point) which returns the translation and rotation of the features, therefore, the relative movement of the vehicle with respect to the previous point. Due to the lack of covariance in the state space, $S^4 = (X, Y, Z\theta)$, this node estimates the covariance using the current velocity of the vehicle. The equation 4.5 describes the influence of the speed with a bias component related to the angular resolution and linear measurement of the lidar.

Local lidar odometry covariance estimation:

$$\sigma_{linear\_min} = translation\_resolution^2 \tag{4.5a}$$

$$Q_b = \frac{\sigma_{linear\_min}}{linear_{max\_threshole} - linear_{min\_threshole}} \tag{4.5b}$$

$$m_x(t) = \dot{X}^2_{local}(t) - Q_b \tag{4.5c}$$

$$m_y(t) = \dot{Y}^2_{local}(t) - Q_b \tag{4.5d}$$

$$m_z(t) = \dot{Z}^2_{local}(t) - Q_b \tag{4.5e}$$

$$\sigma^2_x(t) = |m_x(t) * \sigma^2_x(t-1) + \sigma_{linear\_min}| \tag{4.5f}$$

$$\sigma^2_y(t) = |m_y(t) * \sigma^2_y(t-1) + \sigma_{linear\_min}| \tag{4.5g}$$

$$\sigma^2_z(t) = |m_z(t) * \sigma^2_z(t-1) + \sigma_{linear\_min}| \tag{4.5h}$$

$$\sigma_{angular\_min} = angular\_resolution^2 \tag{4.5i}$$

$$R_b = \frac{\sigma_{angular\_min}}{angular_{max\_threshole} - angular_{min\_threshole}} \tag{4.5j}$$

$$w_\theta(t) = \dot{\theta}^2_{local}(t) - R_b \tag{4.5k}$$

$$\sigma^2_\theta(t) = |w_\theta(t) * \sigma^2_\theta(t-1) + \sigma_{angular\_min}| \tag{4.5l}$$

Where the $linear_{max\_threshole}$, $angular_{max\_threshole}$, $linear_{min\_threshole}$, $angular_{min\_threshole}$ are parameters extracted empirically set to 1 and 0.005 which correspond to the maximum and minimum value to limit and take into consideration the change of the estimated covariance. $m_x(t)$, $m_y(t)$, $m_z(t)$ are the change rate of each linear state (X, Y, Z). and w_θ is the change rate of angular state (θ)

Figure 4.16 shows the performance explained at the beginning of this section. The vehicle starts in the point A (see figure 4.12) and the vehicle finish in the same point doing the U shape.

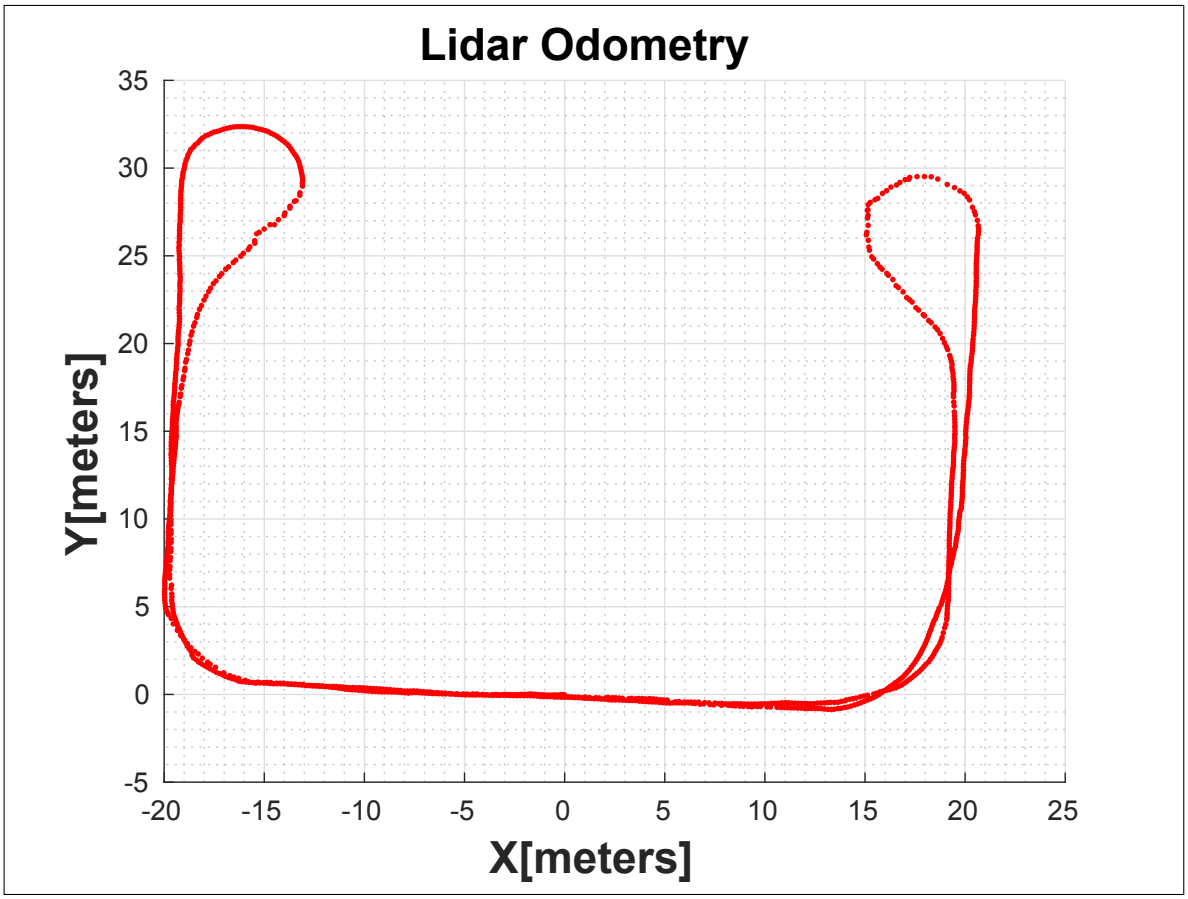


Fig. 4.16 Lidar Odometry outcome

GPS odometry

This section describes the method used in order to calculate the odometry using the GPS. The device used is 3DM GPS U-blox with compass linked to a Pixhawk. The driver responsible to manage and provide the information in the ROS framework is "mavros", an open-source software provided by the company. The configuration is the standard and the calibration was done in the vehicle before fixing the device. The selected position for the device in the vehicle is on the roof aligned with the rear wheel axis and this device provides the latitude, longitude, and altitude along the covariances for each field. The information flows into the ROS framework using the standard messages for any global navigation satellite system using

the WGS 84 reference ellipsoid *NavSatFix* message and the updated frequency for the GPS data information is 3.1Hz.

In order to create the local odometry from this messages, it is necessary to convert it to UTM coordinates with the reference 30T. After the conversion, the UTM coordinates provides X and Y georeference values, hence, it is necessary to subtract the first value of the UTM coordinates to the subsequent readings in order to create the local odometry, where the starting point becomes the local reference (0, 0). It is implemented two approaches for the composition of this odometry. The first one calculates the heading angle in addition of the local X and Y. It uses the previous readings to compute the increments and later extract the slope using the tangent of the two points (see equation 4.6).

Local odometry using GPS readings:

$$x_{raw}(t) = UTM_X(t) - UTM_X(0) \tag{4.6a}$$

$$y_{raw}(t) = UTM_Y(t) - UTM_Y(0) \tag{4.6b}$$

$$local_X(t) = x_{raw}(t) * cos(compass(0)) - y_{raw}(t) * sin(compass(0) \tag{4.6c}$$

$$local_Y(t) = x_{raw}(t) * sin(compass(0)) - y_{raw}(t) * cos(compass(0) \tag{4.6d}$$

$$local_\theta = \tan^{-1}(\frac{local_Y(t) - local_Y(t-1)}{local_X(t) - local_X(t-1)} \tag{4.6e}$$

In order to calculate the covariance for X, Y, and θ, the GPS sensor provides the uncertainty directly based on latitude and longitude transformed to UTM and direct propagation for local X and Y. Regarding the heading angle, the creation fo the heading angle is composed using direct propagation from the previous values of the covariances of X and Y (see equation 4.7).

Compute the covariation for local heading of the GPS odometry:

$$\sigma_\theta(0) = tan^{-1}(\frac{\sigma_Y(0)}{\sigma_X(0)}) \tag{4.7a}$$

$$\Delta\sigma_\theta = tan^{-1}(\frac{\sigma_Y(t) - \sigma_Y(t-1)}{\sigma_X(t) - \sigma_X(t-1)}) \tag{4.7b}$$

$$\sigma_\theta(t) = \frac{\sigma_\theta(0)}{2 + \Delta\sigma_\theta} \tag{4.7c}$$

The second approach does not calculate the heading angle at all and it provides the local X and Y with the direct covariances from the sensor.

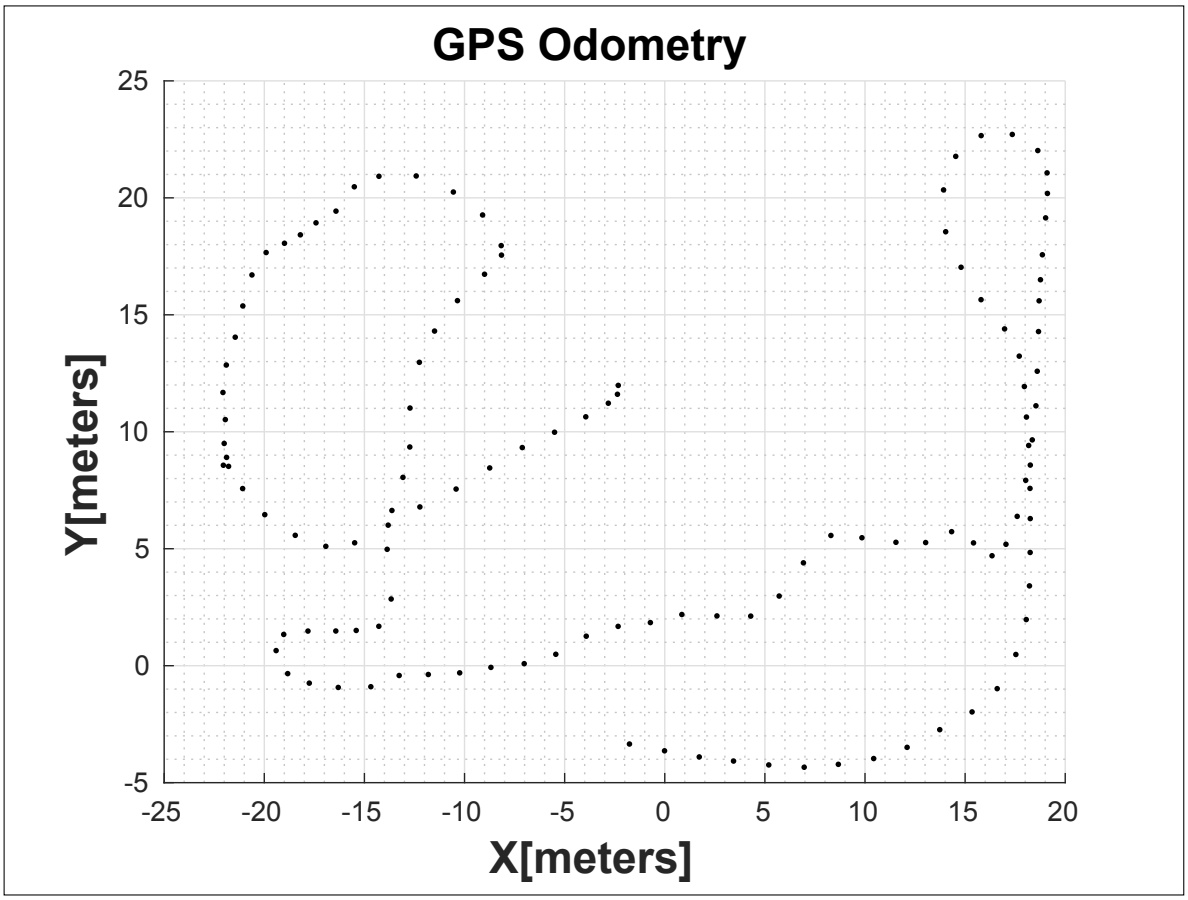


Fig. 4.17 GPS Odometry outcome

4.2.1 Local odometry comparison

This comparison is qualitative due to the fact that is not the main objective of this dissertation. Figure 4.18 shows the figures aforementioned for one lap. It is possible to appreciate the differences between them taking into consideration that the sensors work at a different rate. The best performance after multiple tests and datasets recorded during the time of this project, is done by the lidar odometry (in red), followed by the encoder odometry which has the same shape but the linear distance is not accurate. Visual odometry depends on the quality of the images and it is very sensitive to shadows and illumination. Thanks to the adaptive covariance estimation and the Kalman Filter used, the initial deviation has been corrected but the outcome is not as accurate as the lidar odometry. GPS odometry is very poor due to the building walls near the vehicle that deteriorate the readings and the uncertainty.

After four laps the outcome of lidar odometry and encoder odometry seems constant in shape and distance but encoder odometry is accumulating distance error proved by the

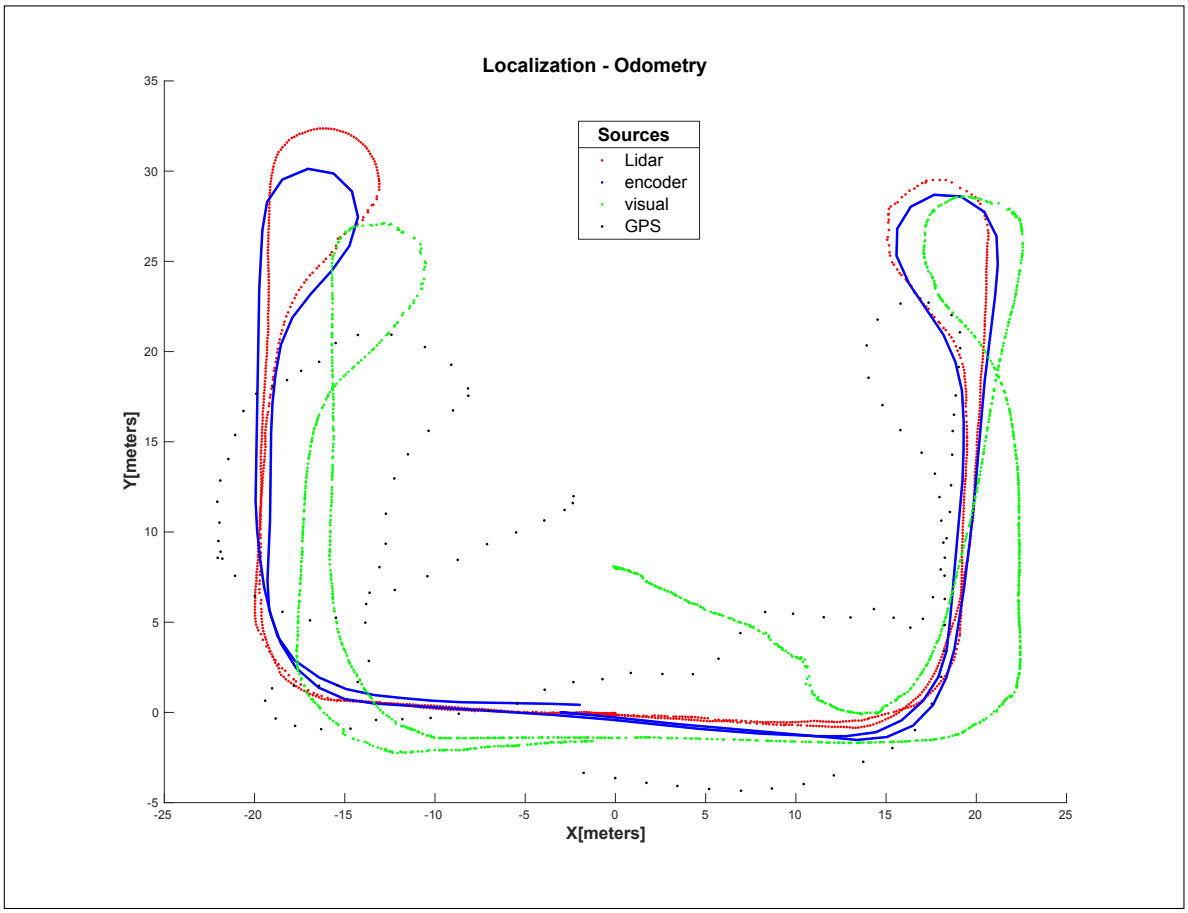


Fig. 4.18 Odometries after one lap.

starting and ending point. Visual odometry fails after one lap due to the first correction and wrong angle calculation in the turnings induced by the shadows and luminosity saturation.

In order to quantify the error percentage of each odometry implemented, the starting point and the final point have been measured, which should be the same by the real performance. The distance covered is estimated by the real dimensions of the building in the straight lines, half of the circumference of the 180 turns. In total in one lap the distance covered by the vehicle is 75.85m and the total of four laps is 303.41m. The study for visual odometry and GPS odometry due to the wrong performance has been neglected for this comparison.

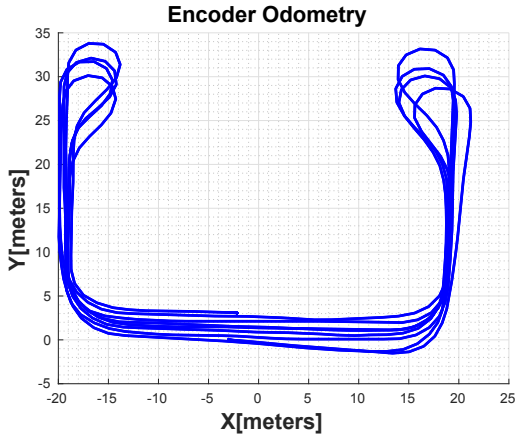


Fig. 4.19 Encoder odometry after four laps.

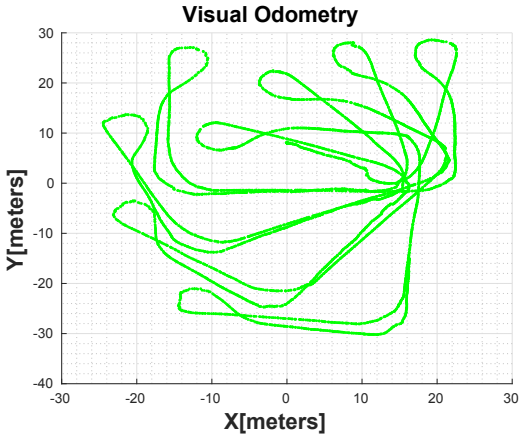


Fig. 4.20 Visual odometry after four laps.

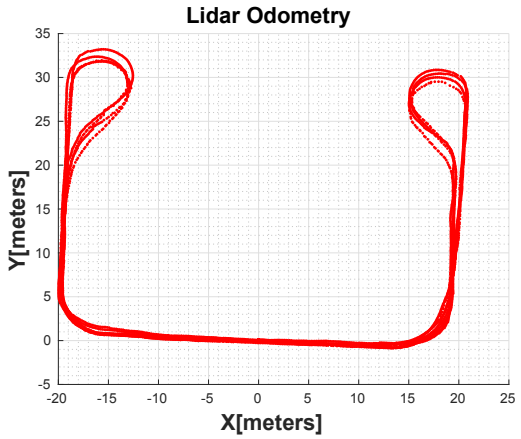


Fig. 4.21 Lidar odometry after four laps.

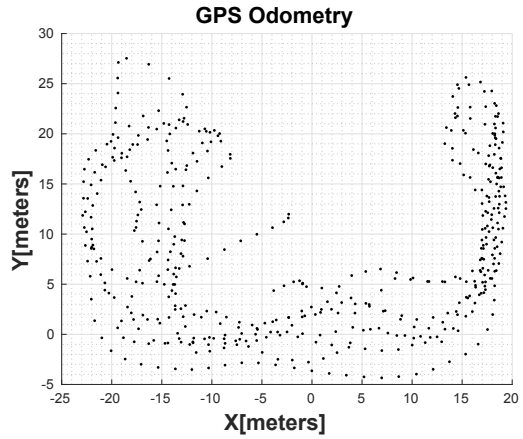


Fig. 4.22 GPS odometry after four laps.

Table 4.2 Odometry methods

Method	Start Point(x,y)	End Point(x,y)	Distance covered [m]	error [m, %]
Encoder	(-3, 0)	(-2.179, 3.08)	303.41	[3.187, 1.04]
Visual	-	-	-	-
Lidar	(-0.03128, -0.0449)	(0.0257, 0.0866)	303.41	[0.1433, 0.47]
GPS	-	-	-	-

4.2.2 Global odometry

It is important for navigation, mapping, and planning, the generation of accurate global odometry because all maps and trajectory will be based on this information, therefore, initial localization is critical. The selected method used in this section for the overall architecture is AMCL (Adaptative Monte Carlo Localization) [26] which uses a particle filter to compare the 2D map (the map is explained in section 4.4) and the 180º laser located at the front of the vehicle. This method computes the estimated initial localization with each iteration and generates new values while the vehicle is moving in order to fit the current position with the global map. This approach is very useful when exists a small accumulated error in the local odometry and the digital localization is shifted from the real localization. In order to understand better this module, the following example is part of the iCab use case which will be extendedly explained in chapter 5.

Figure 4.23 displays the initial position of the vehicle in the global map before starting the AMCL, where the global reference (\map) and the local reference (\odom) are overlapped. The initial guess of orientation and position supplies to AMCL a starting point to initialize the particle filter and compare it to the current readings of the sensor (in this case, the monoplane laser sensor). The particles (between 500 to 2000) are randomly distributed in the neighborhood of the initial guess and after the movement of the vehicle, richer information of the laser scan provides better initial localization, hence, better \odom transformation. The consequence of correcting the initial localization (\odom) transformation is that the vehicle is displayed in the map more accurate and the sensors readings match with the global map static obstacles (figure 4.24). For more information on the AMCL configuration, see Appendix A.2.

Once the initial localization is done, the global odometry is composed using trigonometry relations. In ROS there is a tool called TF (see appendix A.1) which provides the direct relations between the frames. In addition to the vehicle fixed frames, due to the fact that the robot is moving around the campus, it is necessary to define the relations between this movement (\rear_wheel_axis) and the initial point in the local reference (\odom), also called local odometry. Finally, the initial localization of the vehicle provided by AMCL defines the relations between the fixed world frame(\map) with the initial localization of the robot (\odom).

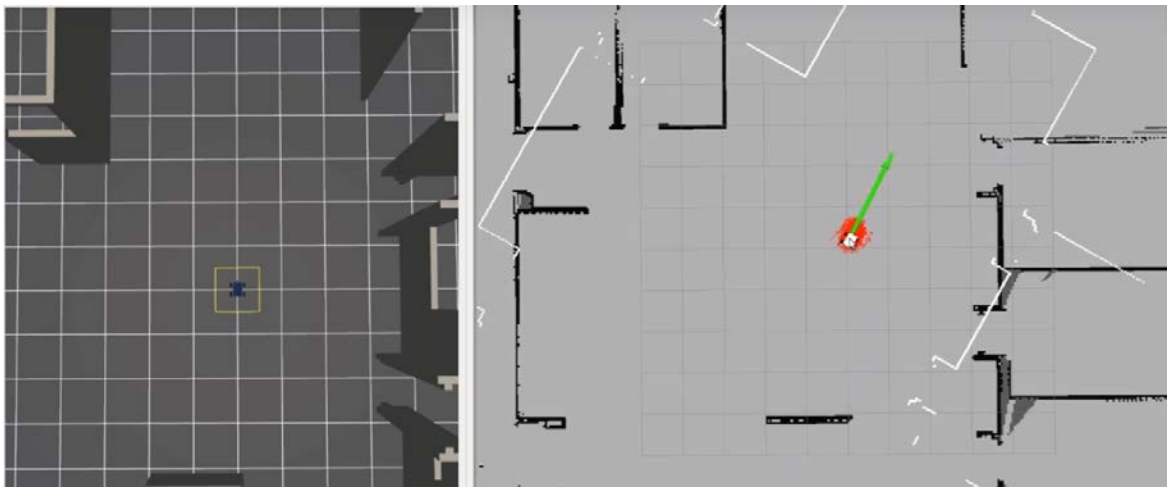


Fig. 4.23 AMCL before the initial guess.

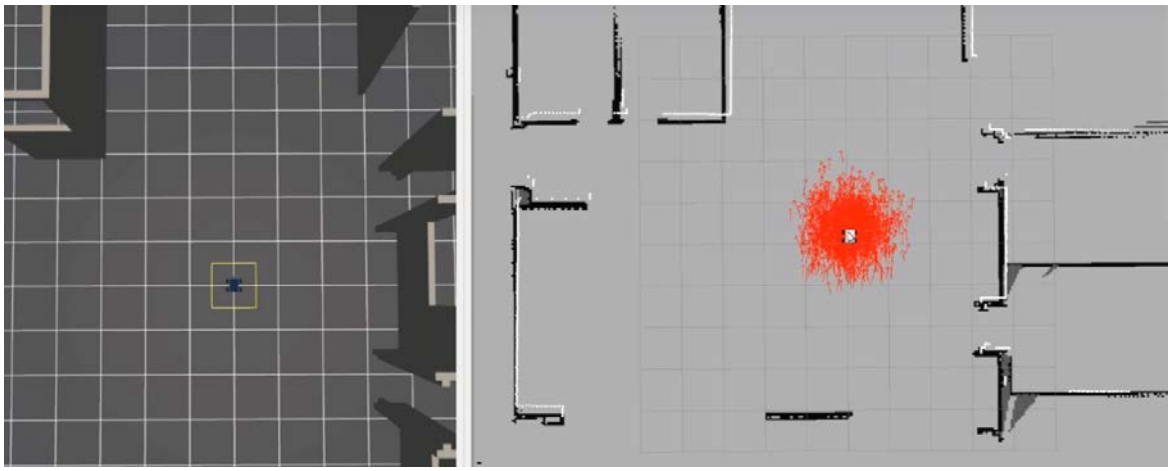


Fig. 4.24 AMCL after the initial guess.

4.3 Perception

This section describes the way the vehicle sense the environment and generates useful information in order to avoid static and dynamic obstacles. The modules involved needs the information of the sensors such as the laser, lidar and stereo camera and the best approach to extract and use this information is the generation of maps, which will be described in the next section.

Elements inside the campus involved in the navigation are static obstacles such as buildings, trees, and lamp post; and dynamic obstacles, the majority, pedestrians. In order to detect them, the use of the stereo camera is crucial for this architecture due to the analysis of the free space navigation for the vehicle. Thus, the work [65] describes how the free space is extracted from the image, then, the obstacles that do not belong to this plane are analyzed and positioned in local coordinates [59] for later be fused with the laser information to enrich the knowledge of the environment. Figure 4.25 describes the sequential steps followed to extract information from the stereo camera. The disparity map composed by rectified images from the sensor is analyzed using the UV-disparity technique in order to extract the free space mask and obstacles mask. The obstacles mask is grouped by elements and marked by ROIs for later extract the local coordinates of each obstacle which do not belong to the free space mask. The local coordinates are calculated using the extrinsic calibration of the camera and the vehicle. Finally, the obstacles are transferred to a grid cell local map and fuse with the laser local map.

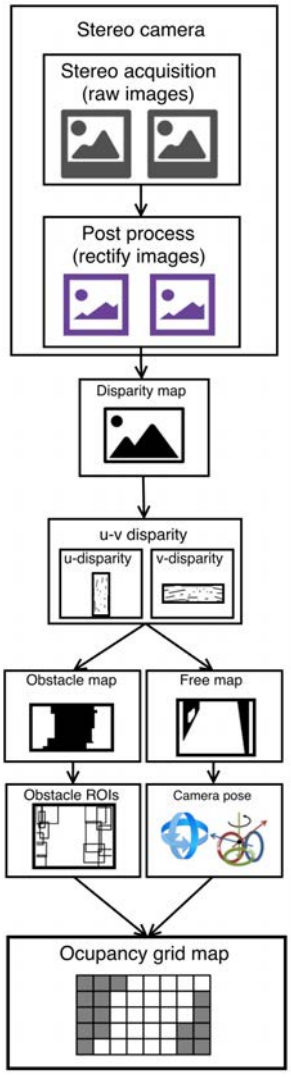


Fig. 4.25 Stereo camera workflow.

Figure 4.26 shows the advantages of using the stereo camera in combination with the laser for obstacles with low height as steps.

Additionally, the analysis of the image provides information about the obstacle classification such as pedestrians and garden using dense masks where each pixel of the ROI is computed in order to separate obstacles from the background. This task is called labeler [8] which group each pixel of the masks by colors of free space, pedestrian, obstacle, and garden 4.28.

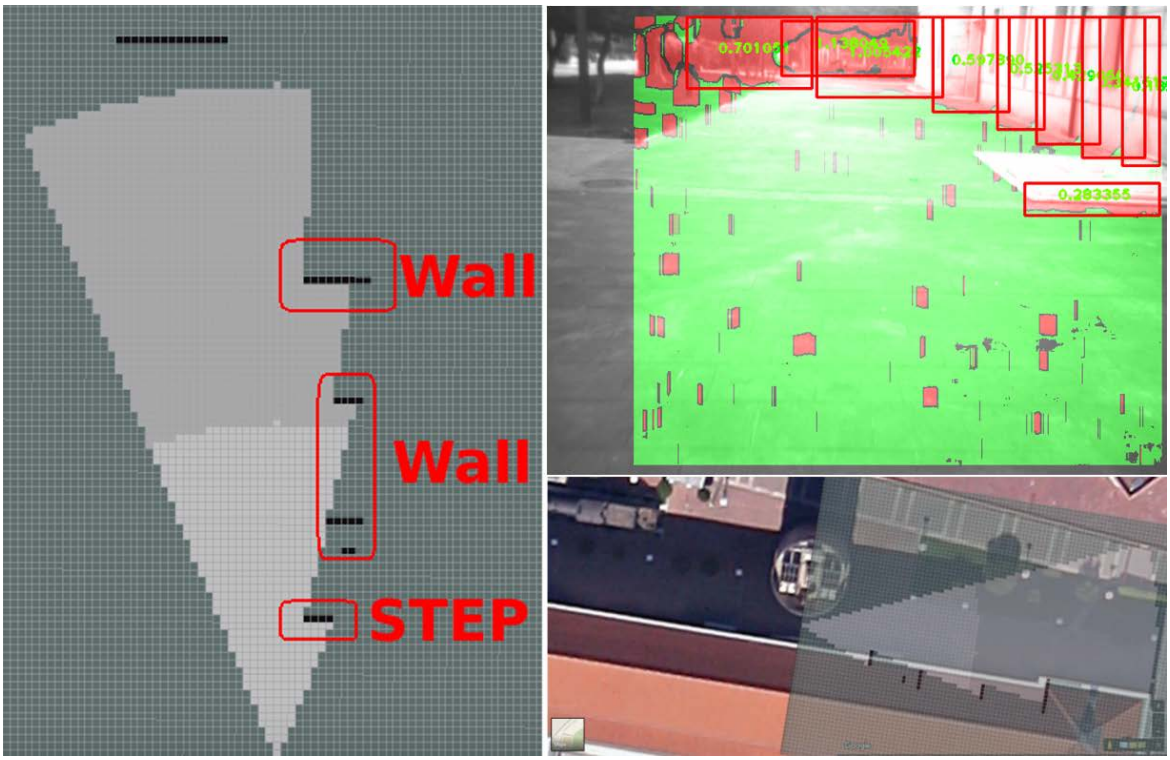


Fig. 4.26 Step detected by the stereo camera and not by the laser. On the left the resulting map after the processing, top right ROIs and free space, top bottom right the experiment zone near Sabatini building.

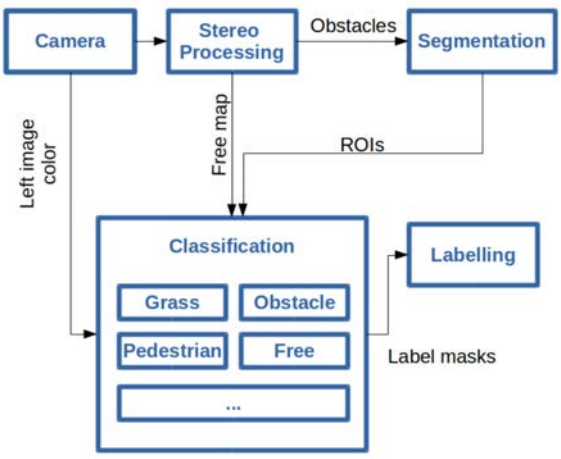


Fig. 4.27 Labelling scheme for detection and classification obstacles, free road, grass and pedestrians [8]

Fig. 4.28 Dense labelling for each pixel regarding to pedestrians in yellow, free space in blue, static obstacles in red and grass in green.

4.4 Mapping

This section describes the ability of the system to provide a useful map based on the sensor information. A map is a collection of grid cells with data gathered from sensors and the fusion layer which describe the environment in terms of free, occupied and unknown. The values are from 0 to 100 to indicate the probability of the cell to be occupied and the value of -1 to indicate that this cell is unknown because there is no information. The size of the map and the resolution are configurable by parameters and after multiple tests, the better resolution for the project inside of the campus is 0.2 m/cell. The mapping section is divided into a global map and local map. This reason is that of the global and local planner that will be described in path planning section 4.5.

4.4.1 Global map

This map contains the static information of the buildings and navigation zones inside of the campus. It has been extracted using the lidar and reconstructing the 3D campus. The procedure used for this task is moving from a starting point in the campus and store the PointCloud for each refresh rate of the lidar sensor in the position of the movement of the vehicle. It has been done a full navigation of the campus and reaches the initial point passing near buildings, trees, lamp post and fountains in order to gather enough information to compose the global map. With the 3D PointCloud of the whole campus, it has removed the floor and projected orthogonal direction to compose the 2D map, where the buildings, lamp post, and trees are in the correct place. Another technique has been tried to create the global map using gmapping with the lidar sensor configured as monoplane of 360 degrees and lidar odometry provided by the odometry manager. In combination with the first approach and due to the impossibility of extract all information for gardens and places where the vehicle is forbidden to pass through, additional information such as boundary limits has been added to this global map. Figure 4.29, 4.30 and 4.31 shows the process of the global map generation.

The main node in charge to load the map and publish as a standard *OccupancyGridMap* message is *map_server* which is a tool provided by the ROS framework for this task. It only needs the dimensions of the map in meters, the resolution in cell size and the origin point of the *backslashmap* frame (reference coordinates [0, 0, 0]) in the grid map. The configuration file is in the appendix A.3 along with other parameters.

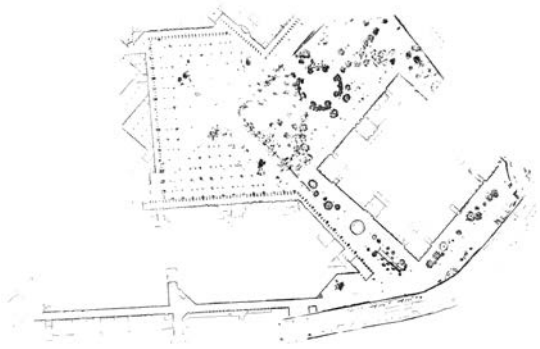


Fig. 4.29 First approach with the lidar 3D reconstruction.

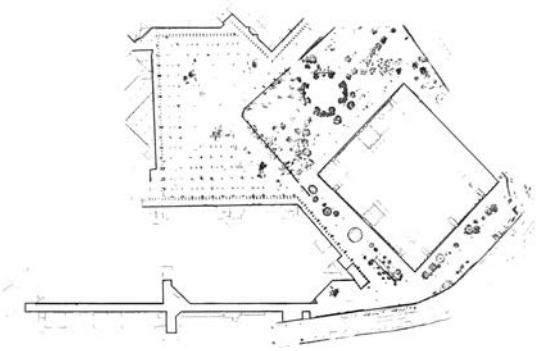


Fig. 4.30 Manually added borderlines.

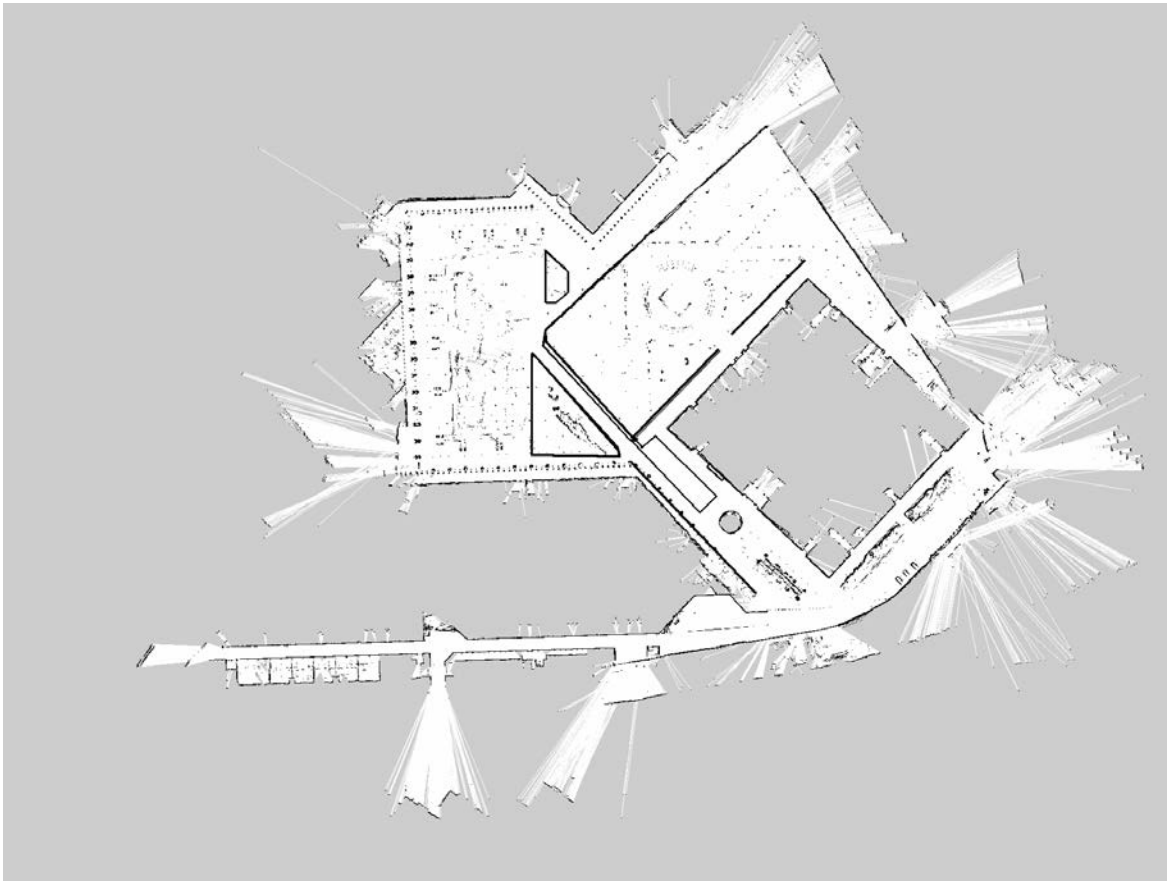


Fig. 4.31 Gmapping, 3D reconstruction and simple boundaries manually added for gardens.

4.4.2 Local map

For local map, the vehicle uses two approaches in order to compare and which one is better. The first one is developed from scratch and very configurable and portable. This approach uses the sensor information independently in order to generate one map for each sensor. Due to the TF configuration, the extraction of extrinsic for each map and the result of combining the information is very simple. Figure 4.32 describes the sensors as input, tested and the combination of all of them into only one local map. It is important to explain that this approach is intended to test, compare and provide maps based on the fusion strategies for research purposes but the use on the iCab project is in its most simple form, where the local maps for each sensor are combined without fusion algorithm.

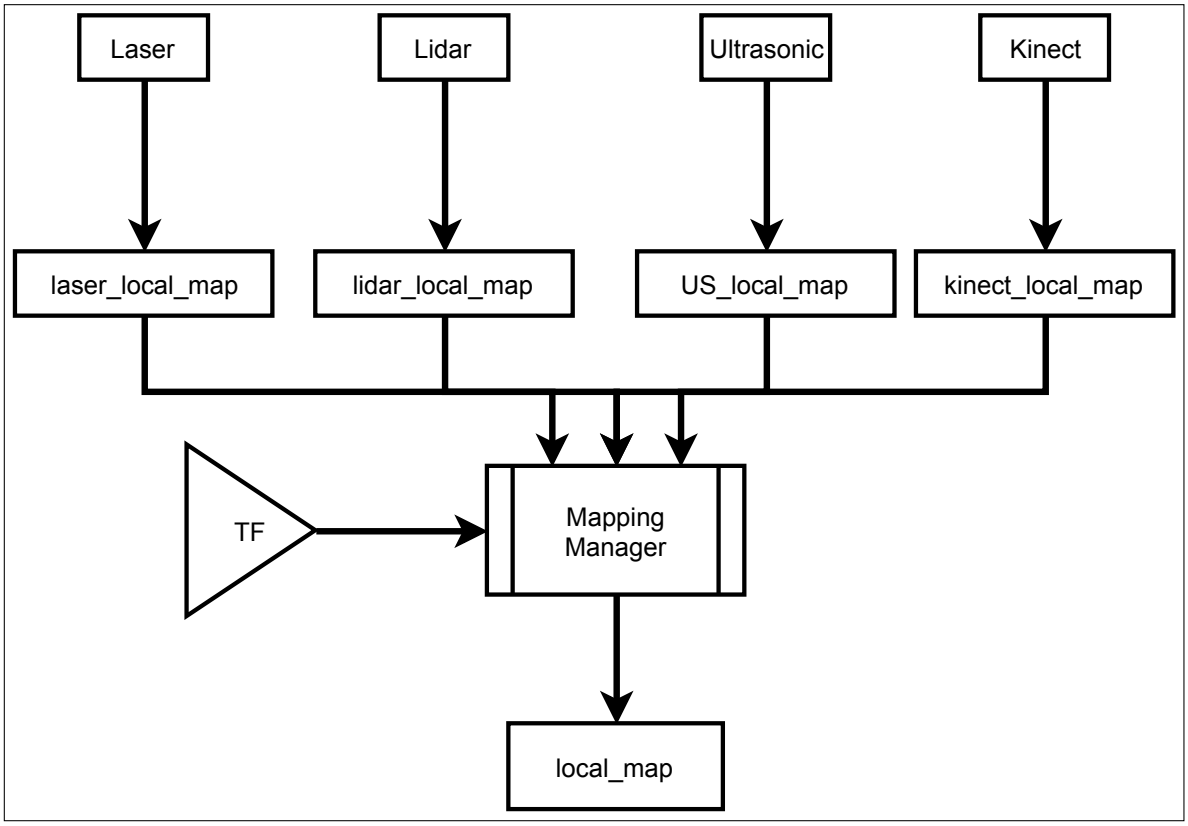


Fig. 4.32 Local map scheme.

laser map

The laser sensor provides standard *LaserScan* messages which are converted to occupancy grid map. The algorithm used to generate the gridmap is simple and based on the sensor values without probability. The reason of using directly the sensor values is the lack of covariance or a previous study of repeatability of the laser beams, however, the outcome of this sensor is high enough to trust in the readings. The technique used is described in algorithm 1 where each cell is started with the value of unknown (-1) and the rest of them are taking the values depending on the laser beams.

Algorithm 1 Laser to map conversion

```
1: for laser beam do
2:     if distance < 80m then
3:         for each cell in between vehicle and value do
4:             cellvalue ← free space                          ▷ value 0
5:     else                                  ▷ equal or greater than 80m
6:         cellvalue ← free space                              ▷ value 0
```

Figure 4.33 shows the readings provided by the monoplane laser. The vehicle is displayed in purple square; figure 4.35 displays the map generated from the laser readings using the algorithm 1; figure 4.34 shows the overlap of these readings and the global grid map with obstacles represented in black (notice the obstacles represented in red at the right of the image, where there were construction fences which were not included as obstacles on global map); Finally, figure 4.36 shows the map overlapped with the global map and the readings.

After the generation of these maps, another tool from ROS framework is Costmap2D which adds information to the occupancy grid map depending on layers. There are three types of layers configured in the system (see the configuration in the appendix A.4).

- **static layer** uses an occupancy grid map message (topic) and it creates a new map with the base information as a copy of the published message. This is the first step to add other layers to the costmap occupancy grid map.
- **obstacle layer** adds the information provided by the sensors to the current map if any. It is used by adding the static or dynamic obstacles in the surroundings. This information should match with the costmap loaded from the static layer if any.
- **inflation layer** adds information of the security perimeter around occupied cells in the map. This technique provides useful information for path planner algorithms in order to stay far from buildings or avoid obstacles with a security distance.

Fig. 4.33 Laser readings.

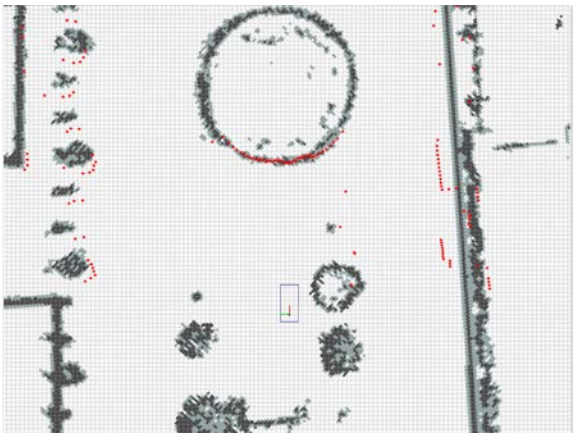


Fig. 4.34 Overlap of the readings and global map.

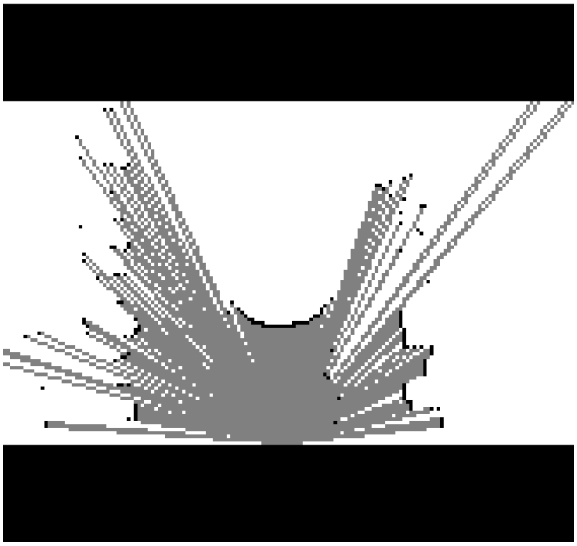


Fig. 4.35 Laser local map generated from map_conversions node.

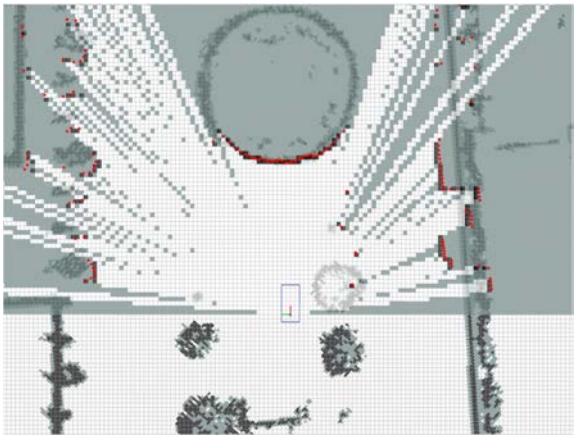


Fig. 4.36 Overlap of the global map, the laser and the laser local map.

For the iCab project, the local map is generated with obstacle layer in addition of inflation layer. This configuration is intended for generating local trajectories far from the buildings, not navigable zones or dynamic obstacles. Figure 4.37 shows the performance of the costmap2D generation where the obstacles are represented in pink, the nonnavigable zone is represented in light blue, the safety limit distance from obstacles are represented in light red and the security distance is represented in dark blue.

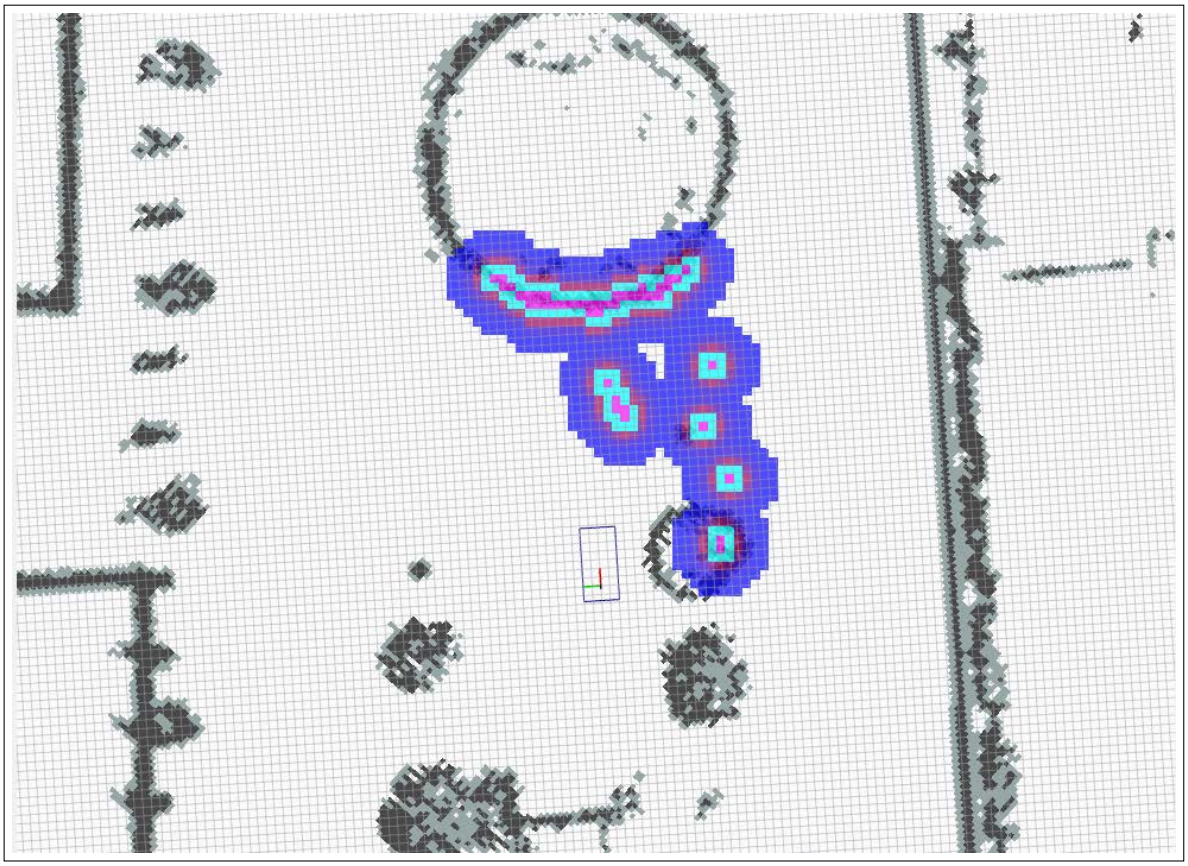


Fig. 4.37 Local costmap performance.

4.5 Path Planning

The path planning module is in charge of managing the trajectories in the environment using maps (with a priori knowledge and current information). As is well known, this module is divided into the *Global Planner* and the *Local Planner*, each of them using different strategies and maps. As said in the previous module, the maps are based on grid cells where each cell has the information of occupied, empty or unknown, therefore, the trajectory generation is based on points with global coordinates.

4.5.1 Global Planner

This submodule is in charge of generating the global trajectory from the initial point, *start point*, to the final point *goal point*. This trajectory is not necessary to be the best but the main requirements are that the waypoints are located in empty cells and far from the occupied cells at a safe distance, and the cells that belong to the direct connection between waypoints are free and navigable. The safety distance is determined by the Inflation layer previously described in section 4.4.

This module is implemented to select different algorithms to generate the trajectory based on the provided maps. The main core of this module is based on a developed ROS package called *move_base* which is intended to manage the movement of the *vehicle*. This module is not ready to work with Ackermann vehicles, therefore, the use of global planner is only for estimation of trajectory purposes.

The method tested and selected due to the fast trajectory generation and performance is Dijkstra. It works even when using a map of big size as the campus (2651 X 2052 with 0.158 m/cell) and is fast enough (. The use of *move_base* package for navigation, allow the system works with A* which is a variant of the Dijkstra method. Figure 4.38 shows the potential values given to each cell in the surroundings of the navigable space for the vehicle. Once the Dijkstra method finds the path, stop generating more potential fields and this is the reason because the map seems incomplete on top of the image. The colors represent the values taken for each cell which increment the value along with the distance of the vehicle. Therefore, more value as the cell is further from the vehicle. The algorithm selects the cells that compose the minimum value from the vehicle to the goal point.

4.5.2 Local Planner

This submodule is necessary to take into account not only the global trajectory (which is not optimized for Ackermann vehicles) but the dynamic obstacles too. The use of two planners

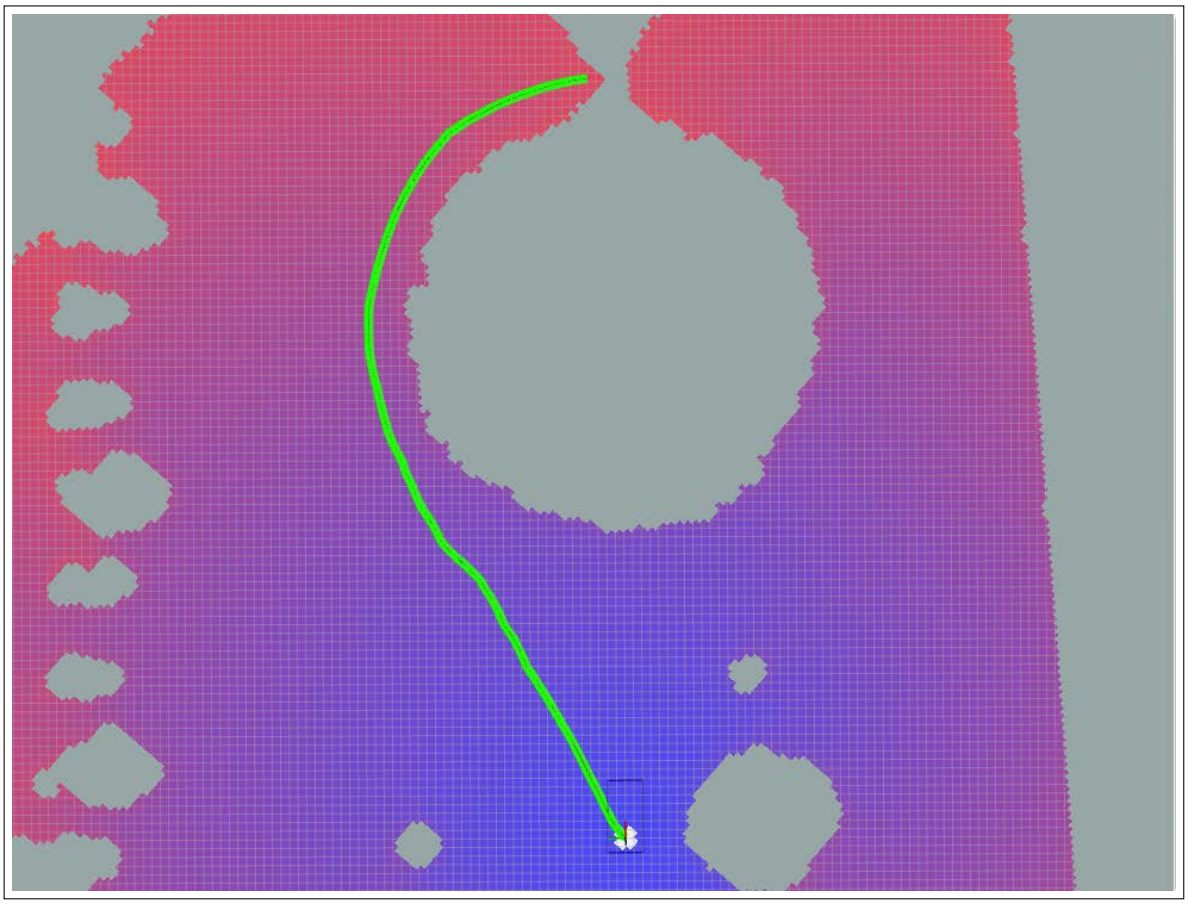


Fig. 4.38 Global trajectory generation from Dijkstra method.

is to save time and resources because the recalculation of the local path is expensive in these two categories. Due to the change of the environment where pedestrians, bicycles, or sometimes animals can cross the campus, it is necessary to recalculate the plan and check the previous trajectory in the event some obstacles enter in the collision path. This recalculation is faster if works in the closest environment of the vehicle and not on the whole campus each time. The reason to save time and resources is that the local map used for local trajectory is really small compared to the global map. For the comparison between them, the local map is the size of 133x133 = 17.689 cells and the global map is the size of 2651x2052 = 5.439.852 cells.

The method used is part of the work of Christoph Rösmann [77] and [78] based on Time Elastic Bands (TEB) for sparse models. The algorithm uses the global plan, the local costmap, and the vehicle velocity to compute the waypoints adapted to Ackermann vehicles. Therefore, the outcome is fast for recalculations and smoother than the global plan.

One drawback for this planner is that the obstacle avoidance for pedestrians is done by the side on which the pedestrians are heading because of the *bands* which deforms part of the global path in the wrong side. Figure 4.39 displays this effect. A complete study for TEB is done in the work [61]. The generation of the local path is based on the local costmap which takes into account the inflation radius of the obstacles and use these values for potential fields to create repulsion forces from the obstacles to deforms the global path.

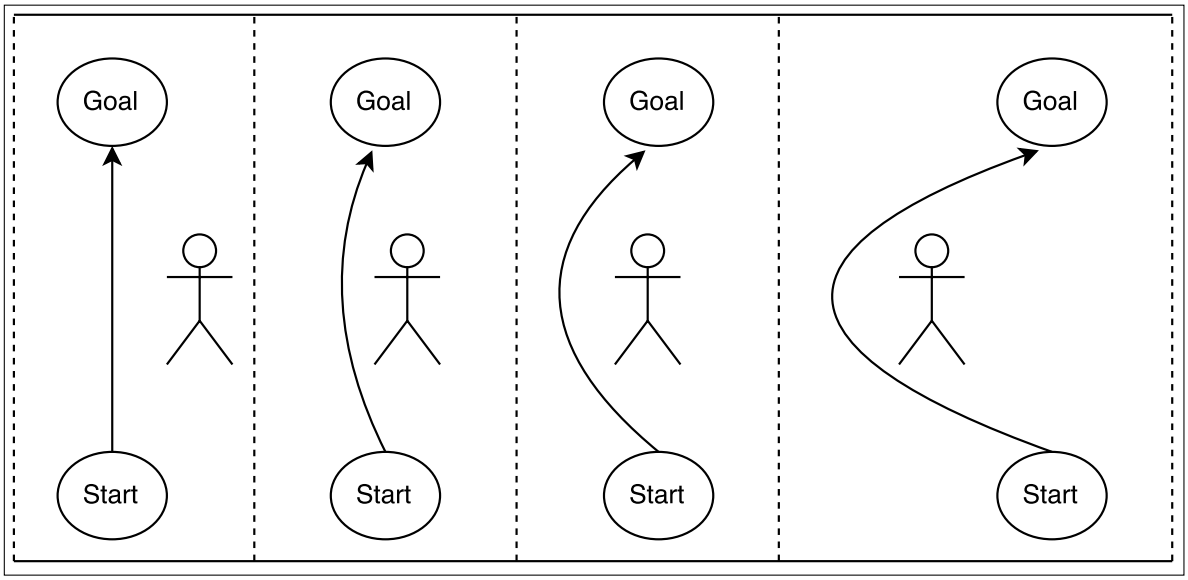


Fig. 4.39 Wrong side pedestrian avoidance.

In figures in 4.40 from left to right and from top to bottom, is displayed the behavior of the planners over a performance of a building avoidance. The sequence of the images shows how the vehicle start from the initial point and the global plan calculates in green the global trajectory. In red, the local trajectory is taken into account only a lookahead distance of the global trajectory from the current position of the vehicle. This lookahead distance determines the longitude of the *band* and the ability to deform it until reaching a maximum in extreme cases. The global planner calculates the global path only once at the beginning of the navigation stage, otherwise, the local plan is recalculating the trajectory between 3.6Hz and 5.7Hz, checking dynamic obstacles and making the path smoother. At the end of the performance, the vehicle has been able to navigate without collisions.

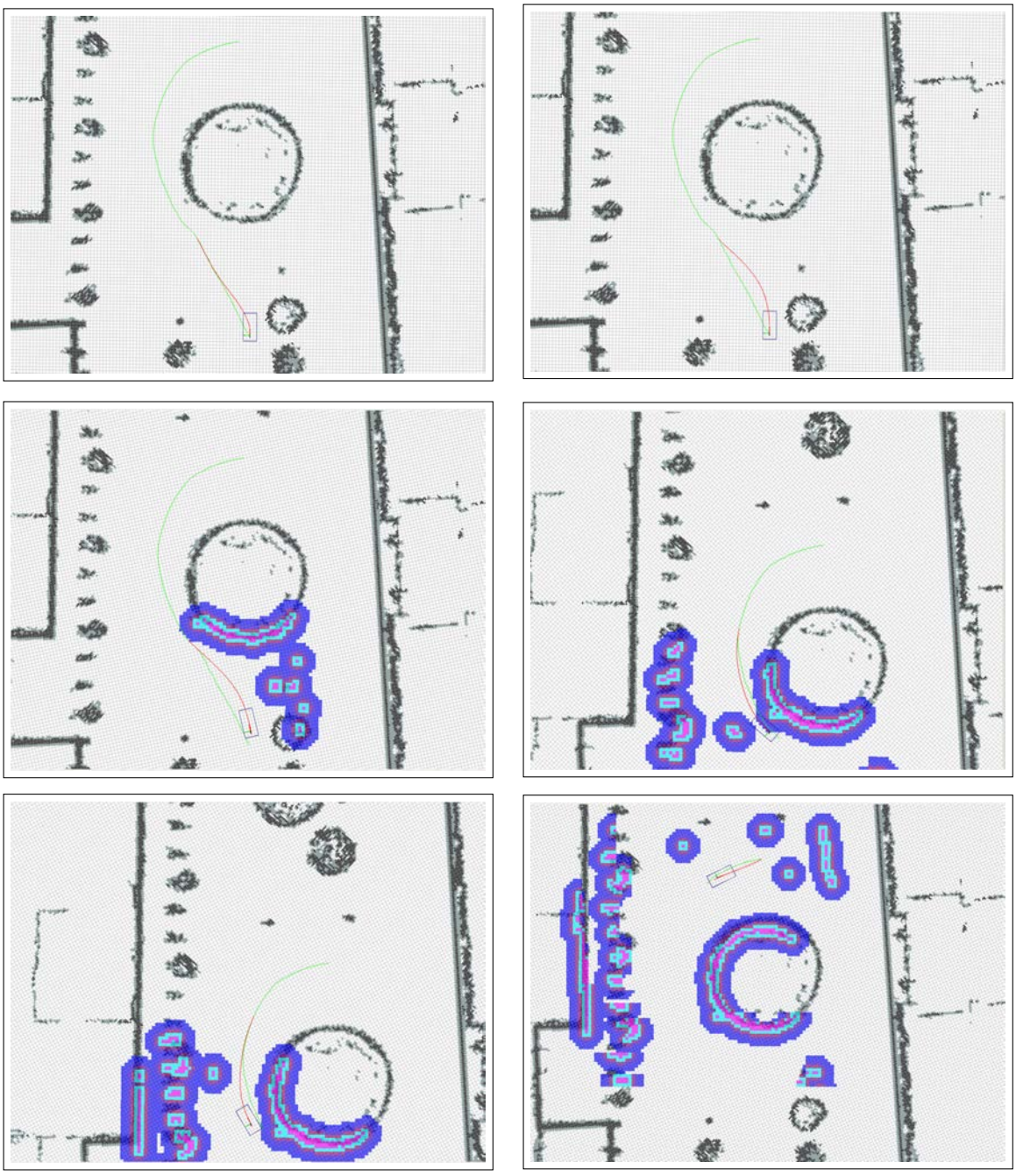


Fig. 4.40 Global and local trajectory generation performance.

4.6 Decision making

Once the vehicle is able to navigate from one point to another without human interaction, the decision-making module takes the control. This module has been created along of the different layers of the architecture which is part of the chapter 5. This module is based on a state machine (figure 4.41) with different states where the vehicle is able to perform various behaviors, including cooperation, automated or autonomous tasks. The developer is responsible for the selection of each task for debugging, testing and comparing purposes but the final objective of this architecture is to configure the vehicle in autonomous mode to perform the iCab project task which is explained in iCab use case, chapter 6.

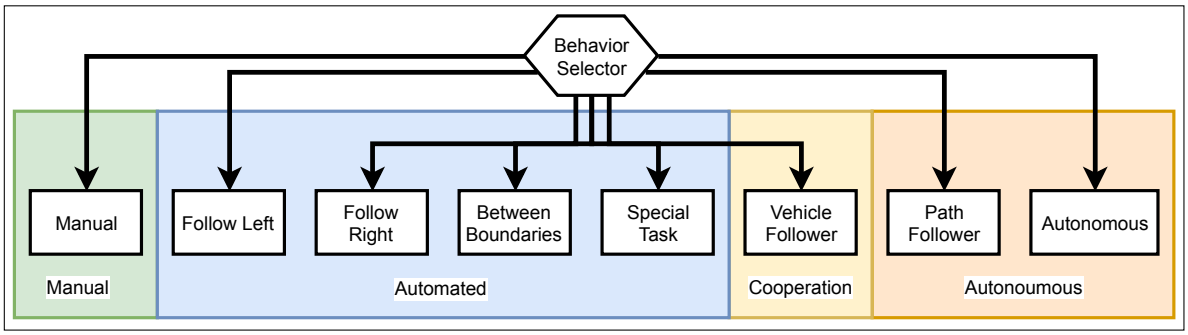


Fig. 4.41 State machine for behavior selection.

Over this module, there is a critical reactive task where the vehicle stops (activate the brake and stop the motor). This task is intended to check the perimeter of the vehicle and in the event of a threat of collision, the commands from the planner are overridden. In the case of this thread disappear, the vehicle continues with the task as usually. For the early version of this module 4.42 is developed a critical stop for elements in front of the vehicle in a distance less than 4m, time enough to stop the vehicle and avoid the collision. One drawback of this version is that will get stuck if navigates close to a wall, tree or lamp post, which are static.

4.7 Graphical user interface - GUI

This GUI represents the interaction and configuration of the vehicle for its different behaviors and displays the internal state of the vehicle for testing and debugging along with visual information for the users of iCab project.

The interface is divided into tabs which shows the relative information of modules such as the iCab project state, the localization, the obstacle detection and mapping and values of battery, drivers and the manual control of the motors.

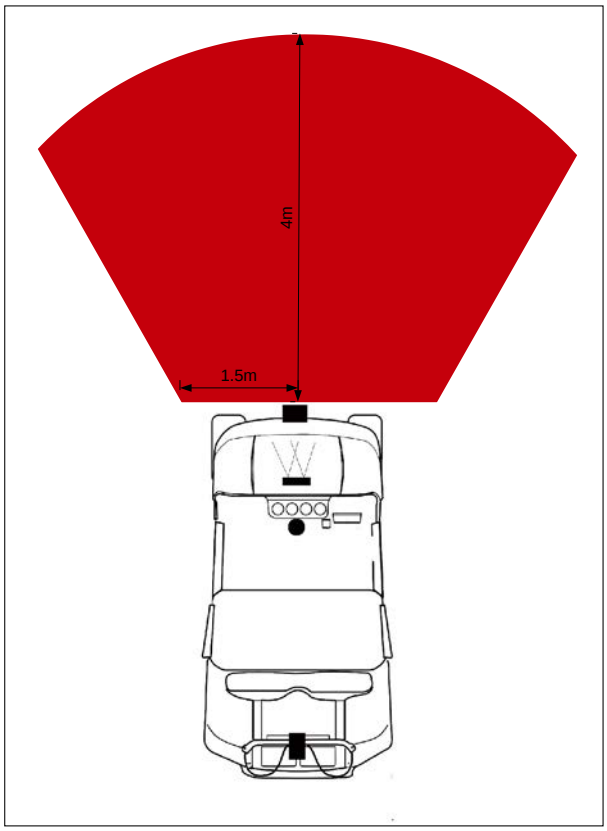


Fig. 4.42 Safety distance in front fo the vehicle to avoid imminent collisions.

Figure 4.43a represents the main tab. On the left part of the GUI, it appears the map of the navigable zone of the campus, which is possible to click to interact with the four Pickup Points of Interest (POI). On the top from left to right, it appears the steering angle in degrees, the PWM provided to the linear motor and the speed in Km/h. After that, it appears a mobile warning motor to activate or deactivate the pedestrian warning [38], orientation information and a button to reset the initial localization. Under this button, it appears the stereo camera left image to display for the user what the vehicle is *seeing*. Under the image, there is the start and stop buttons and the behavior section. This section represents eight behaviors where the vehicle could perform automated movements or could delegate the movement to the user in manual mode. The last button is the continue/pause for automated and autonomous performance. Figure 4.43b shows the second tab called *SLAM* where the local odometry and local mapping are displayed. The third tab 4.43c is for odometry sources, from left to right and from top to bottom: Wheel odometry, GPS odometry, IMU odometry, Visual odometry, Lidar odometry and the fusion odometry. The fourth tab 4.43d is the Obstacles tab where the local map with the vehicle (or vehicles if they are in cooperation mode) appear and the obstacle image showing the free space and obstacles in different colors. The next tab 4.43e is

for manual control, where the vehicle is managed by scrolls bars or selection of values. It also shows the values of self-state of the vehicle such as driver current, battery voltage or encoder values. In the ROS tab 4.43f, the user can configure the core of ROS, the IP, and the hostname. The final tab *extra* is for hidden debug values in case the developers need free space to put variables to display.

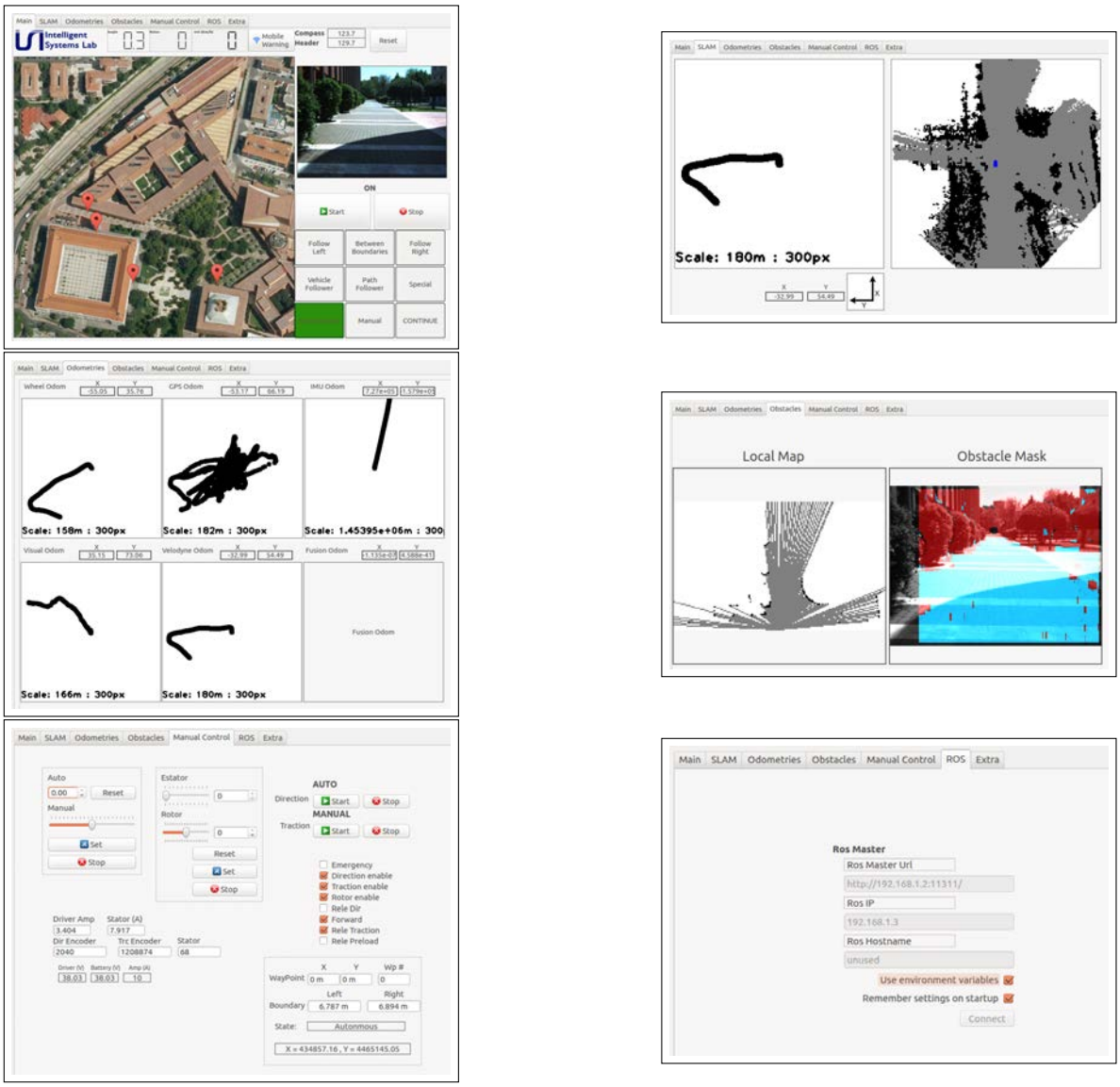


Fig. 4.43 Graphical user interface Tabs.

4.8 Web server

In order to complement the iCab project, the selection of the pickup and drop off points is possible to make it by the Internet. It has developed a web server where the user can request an iCab to pick up in a POI place in the campus and select the drop-off point. This web server is intended not to manage the decisions of which iCab should go to which request, but the purpose is to log the request in a database and visualize the iCab state such as battery level or location in the campus. This module is divided into two submodules, one running in the vehicle as the responsible for accessing to the database with the request and sending the status to the web server and the other running in the workstation located outside of the vehicle responsible for hosting the web server accepting request and showing the iCabs location in the campus map. The work related with this web server is published in [56] shown in figure 4.44.

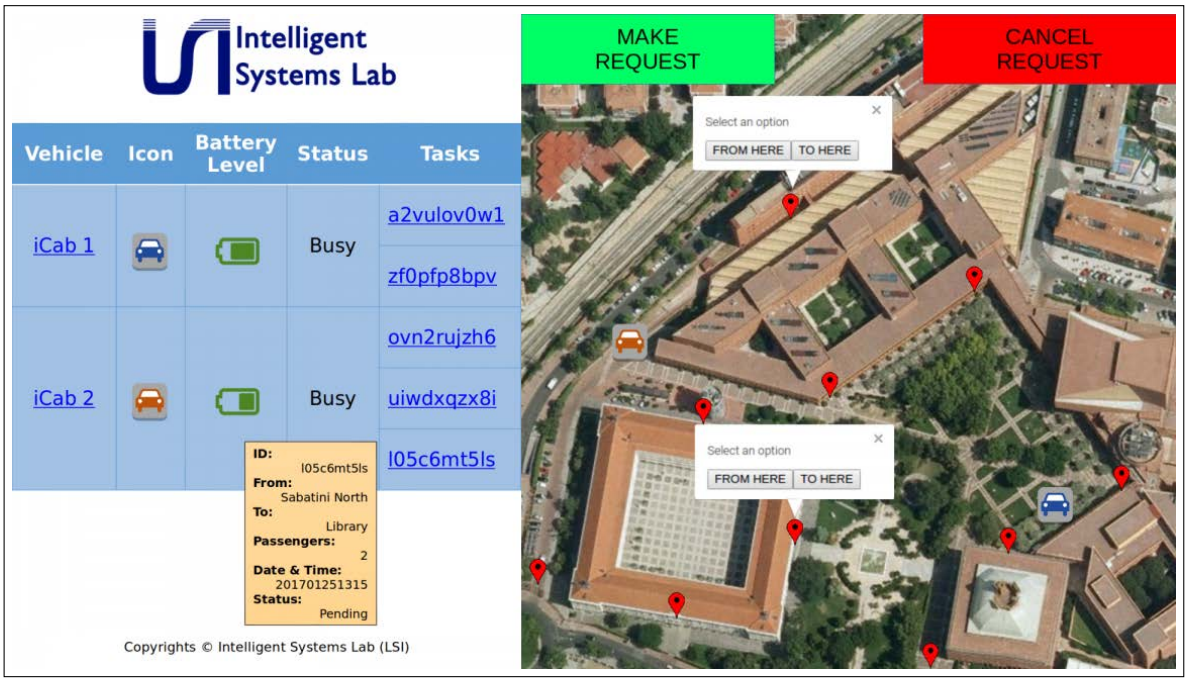


Fig. 4.44 Webserver access from Android mobile phone.

4.9 Communication

This section is part of the work [48] where the configuration of the network is described as a Virtual Private Network (VPN) using a workstation. In the aforementioned document, it was done a study for different configuration via 4G or WiFi inside of the campus. The vehicle to vehicle(V2V) communication is done by 4G with a VPN as the figure 4.45 shows. The use of PVN is intended for security and administration purposes. The communication scheme between vehicles and pedestrian (V2P) is configured by VPN and the application for near pedestrian hazard is studied in [38] shown in figure 4.46.

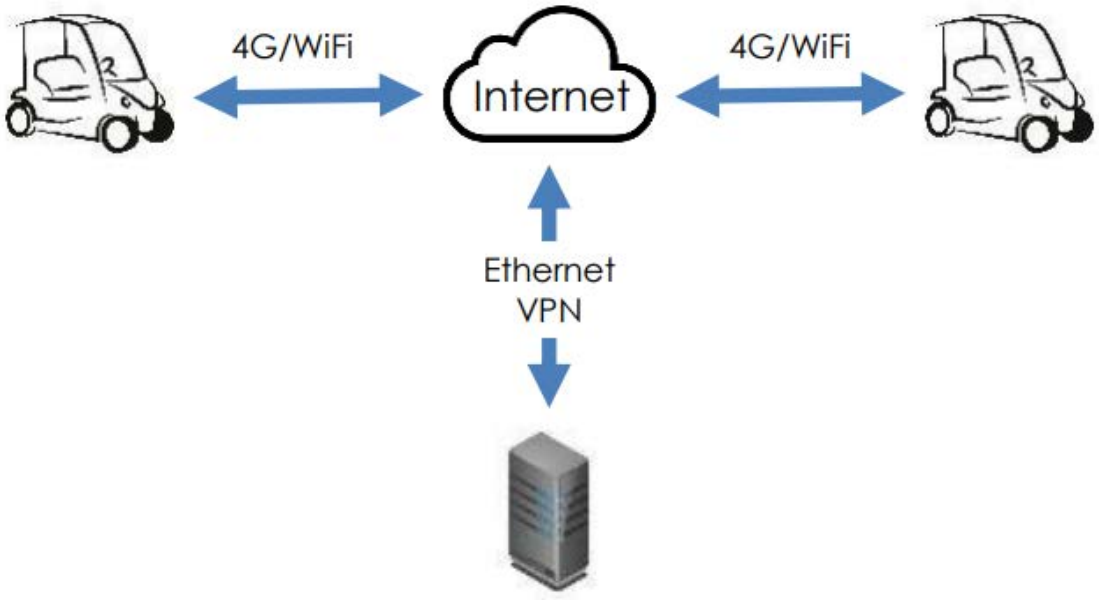


Fig. 4.45 Vehicle to vehicle communication scheme.

4.10 Cooperation

The cooperation module in the iCab utilize the proposed communication schemes to share data among vehicles, pedestrians, and infrastructures, towards the goal of allocating transportations requests to the vehicles as a shared-mobility application. The system is modeled as Multi-robot Task Allocation (MRTA) problem, which is formulated as multi-Travelling Salesmen Problem (mTSP). Various approaches have been proposed to solve the allocation problem in order to reach the best solution based on the set objective function. Through the carried-out experiments and comparative study against several benchmarks, the proposed hybrid optimization-based approach using both Simulated Annealing (SA) and Genetic

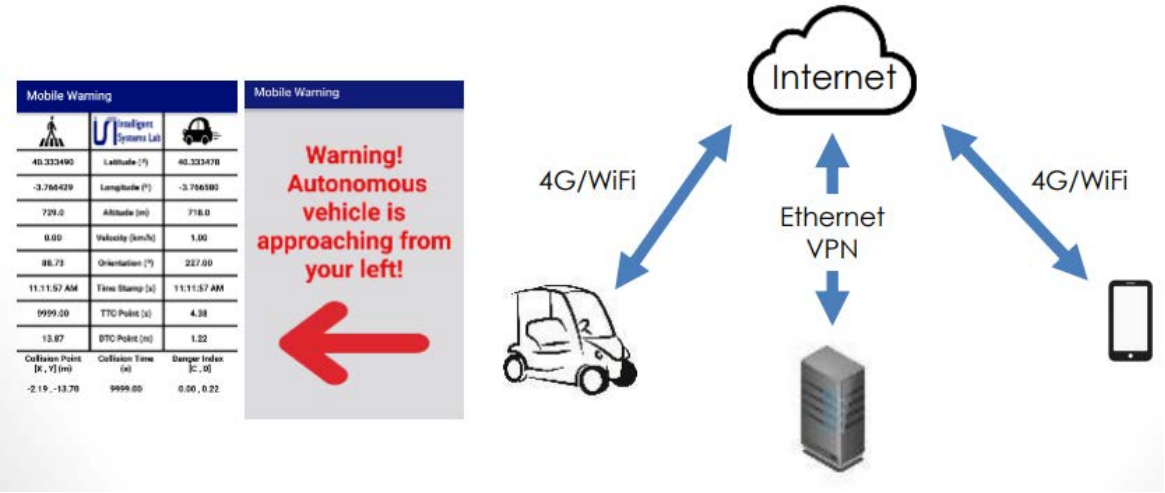


Fig. 4.46 Vehicle to pedestrian communication scheme.

Algorithm (GA) techniques obtained the highest performance and best results. Detailed descriptions of the systems and the proposed approaches are presented in [37].

4.11 Sound manager

This module is intended to interact with the passengers of the vehicle. For the early version, it is another method for debugging purposes and possible problems related to obstacle detection. The utility in this stage is to tell the developers the mode in which the vehicle is configured (by selection in the GUI if it is autonomous or manual mode), and the obstacles in front of the vehicle if there are less than a set threshold. The voices were previously recorded and the sound manager just selects the file to reproduce attending to the vehicle state or the near obstacle detected.

Chapter 5

Software Architecture

5.1 Introduction

The full architecture of the iCab vehicle is described in this chapter with the types of messages used, the connections between nodes and the workflow of the whole system. It is described also the rate of message communication and workflow of the overall shared information.

Along with this chapter, terms as publisher or subscriber will be repeated making reference to nodes which send messages to other nodes or receives messages from other nodes.

The term of *topic* means the unique name of a message, for example, the topic of the map of the whole campus is $\backslash map$ and each node which needs this message should be subscribed to the topic $\backslash map$. There is a nodehandler for each node which creates the XMLRCP information for ROS master in order to be configured properly. This nodehandler is responsible to configure the namespace topics for the messages to publish or subscribe. Another type of communication between packages is *service*, in which a node acts as a server. This server is waiting to another node (client) to make a request (service). These special nodes are used when a task is repeated continuously with different values. For example, the server has the task to calculate the probability using a specific function based on one map. The client sends the service request associated with the value sent to the server and waits for the response of this probability which is calculated in the server. One notorious advantage of using services is the immediate response of the server given the input with a configurable watchdog in case there is something wrong.

Some clarifications need to be specified for ROS framework:

- node: the concept of the node makes reference to each piece of executable code with is inside of the ROS framework using a node name and nodehandle for master discovery.

- topic: each message published by node needs a namespace called *topic* and it is possible to publish under the same namespace *topic* by different nodes along as the message is the same type. There are no limitations about the number of nodes subscribed to a topic.

Due to the critical aspect of sharing messages between nodes, one approach of ROS to make this feature faster is using shared memory and serialization and deserialization in memory. This means that the use of Boost Shared Pointers allows sharing messages between nodes without making a copy of the message. For string messages, there is no difference but working with laser Point Clouds there is a huge difference.

5.2 Core

In order to use ROS framework, it is necessary to run the ROS core (ros master) which acts as the node manager for each program running into the iCab vehicle. This manager is in charge to connect and configure the *tunnels* where the information will flow between nodes as messages. It is possible to appreciate the workflow of the ROS framework in figure 5.1 where the core gather the information of each node (previously provided by each node using XML/RCP) and make the connection between them. Once the core receives this information, using master discovery (which detect each node in the same network), it determines what the node wants to do, if publish, subscribe or act as service. Later, roscore generates the connection between nodes configured by TCP (acknowledge of the message received).

5.3 Standard Messages

One of the most difficult decisions in the software architecture is the use of custom messages or standard messages.

Using custom messages require less amount of work in software development of each node because the developer uses only one message, subscribe to it and if needs more information, just edit the custom message adding new information. This procedure is wrong in several aspects, for example, the data recorded with old custom messages is obsolete when one custom message recorded changes and is not possible to work with the recorded data. Another problem is sharing recorded data with other research groups because you have to share your custom data too. Therefore, using standard messages not only makes the recorded data always available, it helps to test multiple algorithms that use similar data from other researcher groups and increase the project life cycle.

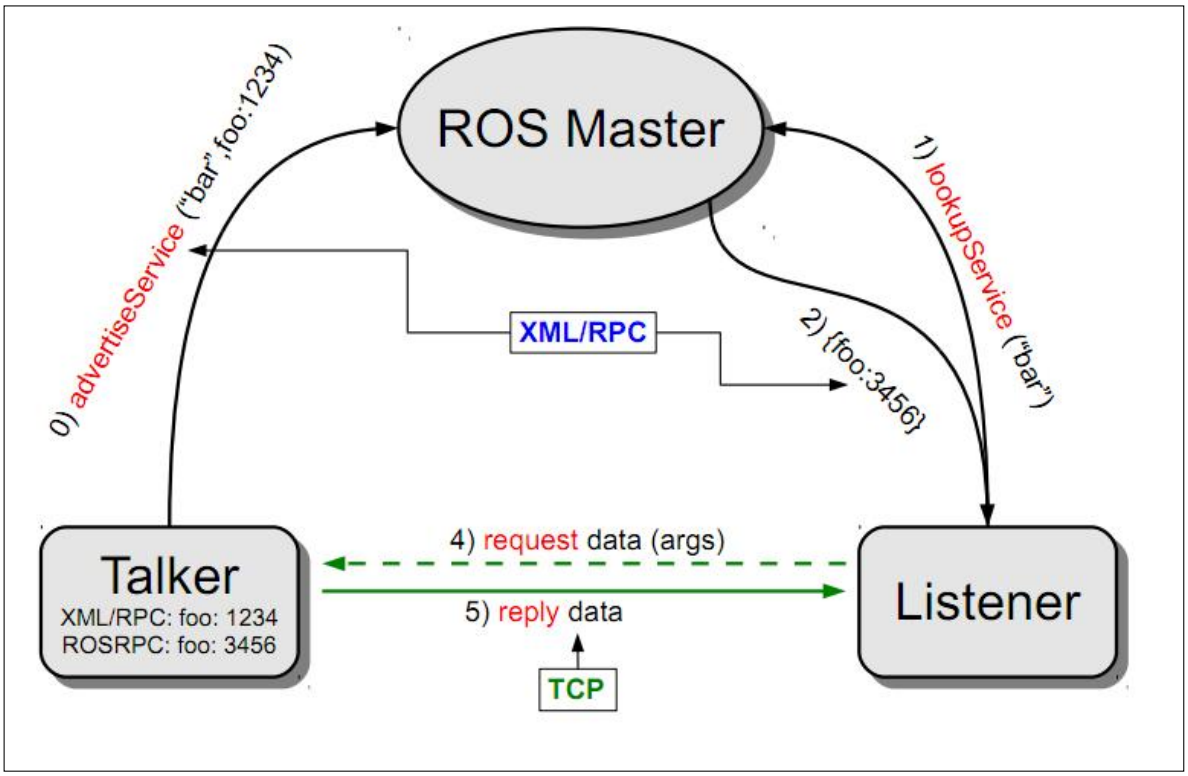


Fig. 5.1 State machine for behavior selection [73].

Along with this thesis, it is necessary to remark the importance of the use of standard messages for the fields related to each module. These standard messages are classified depending on their purpose described in figure 5.2. Notice that the name of the first group is standard messages that describe only the basic messages for programming purposes but all groups displayed in the figure are standard and ROS compliant messages that have been used by ROS community which has been discussed and created by Special Interest Groups (SIG) composed by a group of experts around the world in each field. Some examples are related to perception, sensor data acquisition, and navigation.

In order to synchronize, record and simulate messages in ROS frame environment, there is another field called *Header* which contains sensible information about the frame publisher of the message and the time stamp when the message was published. The specific fields described in the header are:

- seq: Number of consecutive messages published under the same topic.
- stamp: Time stamp divided into two double variables called seconds and nanoseconds
- frame_id: Frame where the messages have been published. It is related with the TF node (see appendix A.1)

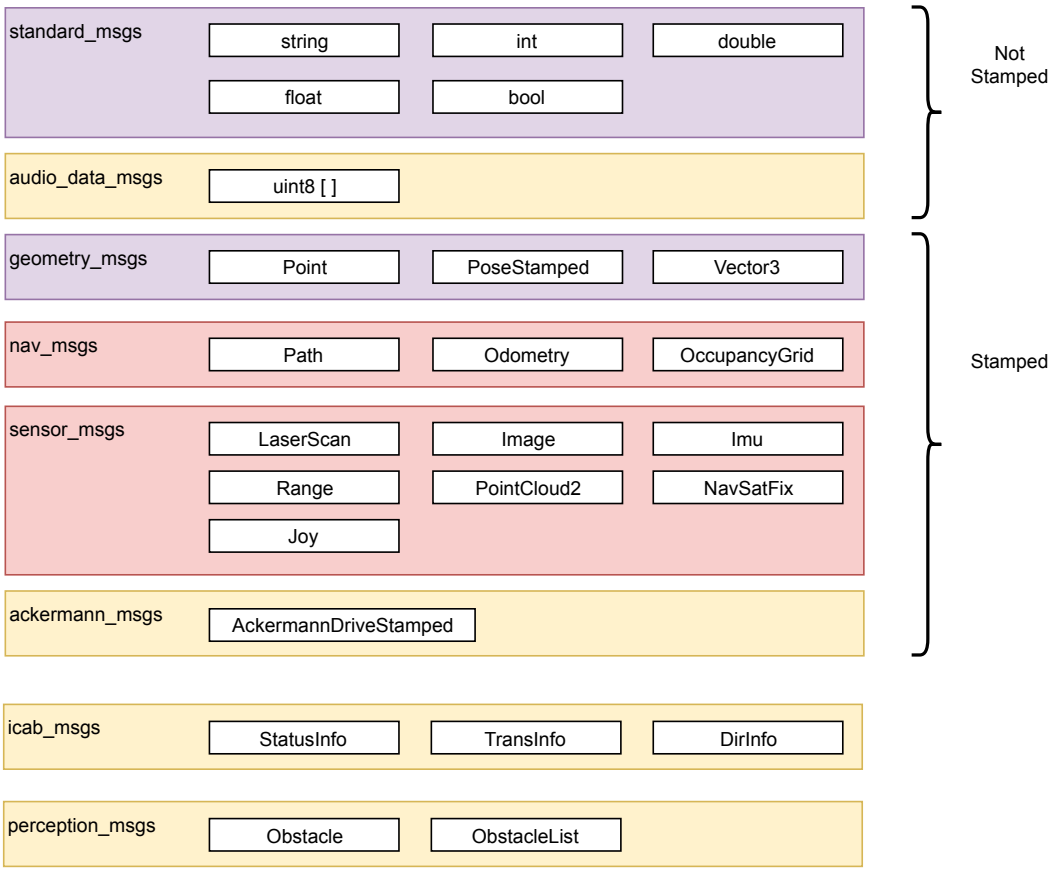


Fig. 5.2 ROS standard messages used in iCab architecture.

5.3.1 Sensors

This section describes the concept of HAL (Hardware Abstraction Layer) where the sensors used are not important neither the features or spectrum meanwhile the messages provided by these sensors are compliant and coherence with the architecture. In this architecture, there is no real HAL but the performance is similar due to the work with standard messages. Therefore, the use of these messages are required upstream and could be provided by the real sensors, recorded data files or simulated sensors without any modification in the architecture. This performance is ideal to work in a research platform as the iCab due to the constant tests with recorded data and enable the extension in the life cycle of the overall project and continuous evolution in software and hardware components.

Each sensor needs a driver enabler to work with ROS framework. The sensor data is acquired using a program that communicates directly with the physical sensor and later publishes the data as standard messages. For each sensor in the vehicle, it has been used the sensors messages described in 5.2, thus, for the laser monoplane it has been used the message *LaserScan* which contains not only the distance and intensity measured for each

laser beam in a vector but information related with the device itself and its configuration such as minimum and maximum angle, angle resolution, maximum and minimum range distance. For the stereo camera, there is published two *Image* messages at the same time containing the image itself in a vector of the uint8 type where the image matrix is stored and height, width, encoding, step and if is big-endian information. With this metadata is possible to convert to OpenCV and analyze and work easier. The IMU information is published by *Imu* message where the complex type of messages are nested such as the orientation, angular velocity and linear acceleration which are types of *Quaternion*, *Vector3* and *Vector3* in this order with each associated covariance for each measurement in the form of a float vector of nine elements each. The Ultrasonic sensor provides *Range* messages with the range in meters, min and max range in two variables, field of view and radiation type (0 for Ultrasound and 1 for Infrared). Lidar and Kinect provide *PointCloud2* messages with the information of the type of data Points, height, width if it is big-endian, the length of a point in bytes and the length of a row in bytes and the existence of invalid points when the pointCloud is not dense. Along of these sensors, one sensor wide forgotten is the Joystick which type of message is *Joy* describing the axes measurements and the buttons in two vectors for each field. Finally, the message related with the Global Positioning System is *navSatFix* containing the status of the satellite, latitude, longitude, altitude, covariance for each field, and the type of this covariance.

5.3.2 Actuators

In the case of the output of the system, the final commands generated for the motors are the type of *AckermannDriveStamped* which contains critical fields such as speed, acceleration, jerk, steering angle and the change rate of steering angle. For the speakers, the messages used are the type of *AudioData* with a plain vector of uint8 which are transformed using GStreamer libraries upstream in the architecture to send to the speakers.

It is necessary to clarify the output/input for the brake and motors. In the case of iCab project, it is used for the low-level control custom messages. The use of this messages is related only with the iCab platform due to the hardware configuration in order to access and control each motor and retrieve information of the encoders. It has been said that the use of custom messages are not recommended but at some point, they are necessary to control the specific hardware. The definition of this custom messages in a package called *icab_msgs* are displayed in figure 5.3 which shows the nested messages with the description of each field used.

The main data structure is *StatusInfo* described in 5.4 where all information related with the vehicle is gathered only for get the information, not control. Notice that inside *StatusInfo*

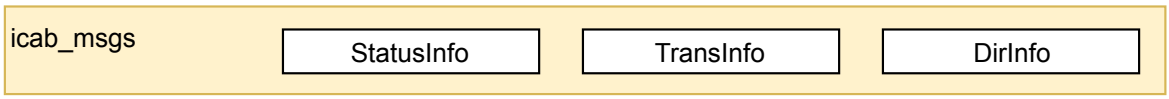


Fig. 5.3 Custom messages for iCab controller.

there is two types, *TransInfo* and *DirInfo* described in figures 5.5 and 5.6 used for send the commands to the driver to control the vehicle. Each of them has included a *Header* type for synchronization purposes.

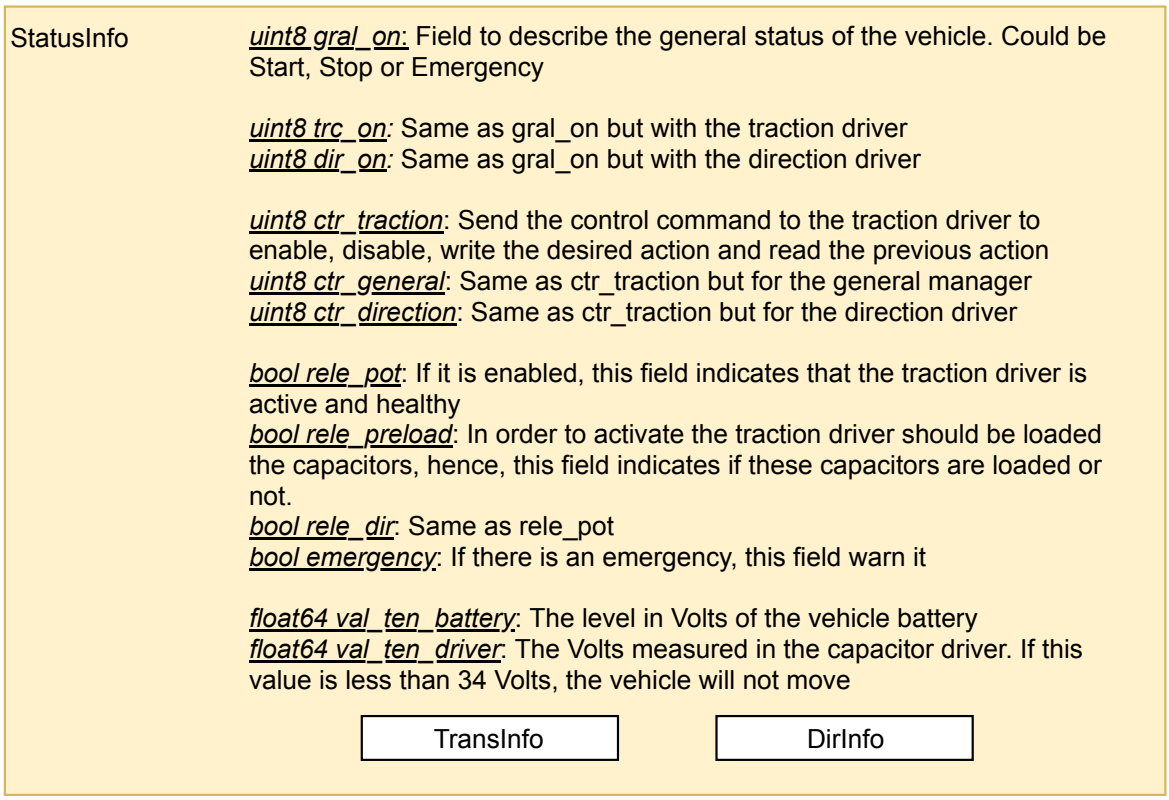


Fig. 5.4 Custom message container to gather all information of the vehicle driver and status.

5.3.3 Perception messages

Along with this messages, there are perception messages used in the architecture to describe the elements detected in 2D, 3D, and segmentation of them as well. These messages have been developed in the Intelligent Systems Laboratory since 2015 passing through different stages until finally gather the required information to share upstream in the architecture. The necessity of customization of these messages appears when there are no standard messages in ROS repositories to work at the same time with 2D, 3D, and classification by semantic

TransInfo	*bool tract_enable*: Enable or disable the stator *bool rotor_enable*: Indicates if the rotor driver is enabled *bool forward*: Indicates the direction of the desired movement *int64 consigna_exc*: Specify the desired PWM from 0 to 100 to feed the stator *int64 consigna_rot*: Specify the desired PWM from 0 to 100 to feed the rotor *float64 velocity*: Current speed of the vehicle in m/s *float64 cte_exc*: Constant bias for stator *float64 cte_rot*: Constant bias for rotor *float64 val_amp_exc*: Measure the instantateous Amperes demanded by the stator *float64 val_amp_driver*: Measure the instantateous Amperes demanded by the driver *float64 consigna_exc_rpa*: Limit the acceleration of the PWM change in stator *float64 consigna_rot_rpa*: Limit the acceleration of the PWM change in rotor *float64 pid_kp*: PID values for position control *float64 pid_ki*: PID values for position control *float64 pid_kd*: PID values for position control *float64 pid_error*: PID values for position control *float64 pid_value*: PID values for position control *int64 val_encoder*: Encoder value of the traction motor

Fig. 5.5 Custom message for Translation driver status and control.

DirInfo	float64 angle: Desired steering angle float64 encoder: Absolute encoder value for steering angle float64 val_amp: Instantaneos Amperes demanded by direction driver

Fig. 5.6 Custom message for Direction driver status and control.

methods. Figure 5.7 displays the structure and fields of these messages and the importance of each field to share critical information for later processing and including into the maps.

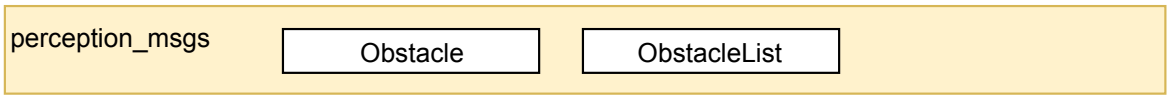


Fig. 5.7 Custom messages for Perception modules.

Obstacle	*uint32 id*: Identifier of the obstacle (for tracking purposes) *string kind_name*: The name of the class of the obstacle *int8 kind*: The ID of the class of the obstacle *float32 score:* Conficence for the object belonging to the class "kind" (given at classification stage) *int8 truncated*: Truncated value *int8 occluded*: Occluded value *int8 disp_min*: Minimum disparity when detected *int8 disp_max*: Maximum disparity when detected *int8 elevated*: If it is over the floor *float32 height*: Size in meters *float32 width*: Size in meters *float32 length*: Size in meters *sensor_msgs/RegionOfInterest bbox*: For blob labeling (Bounding box in the image plane) *geometry_msgs/Point32 location*: 3D Location in space coordinates *float32 alpha*: Viewpoint of the obstacle *float32 yaw*: Rotation around the obstacle's vertical axis *geometry_msgs/Twist velocity*: Velocity in m/s or pixels/s in the world or in the image sequence

Fig. 5.8 Custom message for obstacle 2D, 3D and semantic classification.

Id	Category	R	G	B
0	**Background**	**0**	**0**	**0**
1	Ground	81	0	81
2	Dynamic	111	74	0
3	Static	0	0	0
10	**Vehicle**	**0**	**0**	**255**
11	Car	0	0	142
12	Van	0	0	110
13	Truck	0	0	70
14	Bus	0	60	100
15	Caravan	0	0	90
20	**Person**	**255**	**0**	**0**
21	Pedestrian	255	20	40
22	Person sitting	180	20	40
23	Rider	255	100	0
30	**Two wheelers**	**255**	**0**	**255**
31	Cyclist	255	100	100
32	Bicycle	119	11	32
33	Motorcycle	0	0	230
40	**Other vehicles**	**0**	**0**	**110**
41	Trailer	0	0	110
50	**On Rails**	**0**	**175**	**150**
51	Tram	0	175	140
52	Train	0	175	120
60	**Flat**	**128**	**0**	**128**
61	Road	128	64	128
62	Sidewalk	244	35	232
63	Parking	250	170	160
64	Rail track	230	150	140
70	**Construction**	**150**	**150**	**150**
71	Building	70	70	70
72	Wall	102	102	156
73	Fence	190	153	153
74	Guard rail	180	165	180
75	Bridge	150	100	100
76	Tunnel	150	120	90
80	**Nature**	**120**	**170**	**90**
81	Vegetation	107	142	35
82	Terrain	152	251	152
83	Sky	70	130	180
90	**Object**	**153**	**153**	**153**
91	Pole	153	153	153
92	Pole group	153	153	153
100	**Signs**	**255**	**255**	**0**
101	Generic traffic sign	220	220	0
102-189	Traffic signs	220	220	0
190	Traffic light	250	170	30
200	**Lane markings**	**255**	**255**	**255**

Fig. 5.9 List of type of obstacles in the field "kind" mix of KITTI and Cityscapes guide lines.

ObstacleThreat	*bool visual_obstacle_detected*: If there is an obstacle in range detected from camera *bool laser_obstacle_detected*: If there is an obstacle in range detected from laser *float64 visual_obstacle_distance*: Distance associated if any *float64 laser_obstacle_distance*: Distance associated if any *float64 minimum_index*: If there is more than one obstacle, the index of the closest one *float64 minimum_distance*: If there is more than one obstacle, the distance of the closest one

Fig. 5.10 Custom message to consider obstacles as threats and the information required for the reactive layer to stop the vehicle.

5.3.4 Navigation messages

The navigation-related messages includes *Path*, *Odometry* and *OccupancyGrid* which describes trajectories, odometries and occupancy grid maps previously described. These messages are standard from ROS repositories and are widely extended in the navigation community. Modules such as path planning and mapping are directly related to these messages and every module linked to them work with standard messages to enhance and improve the interchangeability of modules. For example, in this iCab project, has been used different types of map generators which work perfectly with the mapping server of ROS repositories and Costmap2D.

5.4 Architecture workflow

Each module previously described is represented by a figure in this section where the aim of the module could be to create a high-level information such as maps or trajectories and other module use this high-level information to generate the action associated. The colors in figures take high importance because of the necessity to distinguish the creation and use of high-level information such as maps, trajectories, and odometries.

5.4.1 Mapping workflow

Figure 5.11 represents the nodes involved in the creation of the maps used in the architecture. It is divided into local mapping and global mapping where the creation of three different maps is necessary to share with localization and planning module. The local map is generated from lidar, laser and obstacle information. As said before, the lidar and laser information uses PCL and LaserScan messages. From the perception module, the obstacles are detected and classified into a list with 2D and 3D information. This list is fused with lidar and laser information in order to create an auxiliary gridmap. The algorithm used for this fusion, at the moment of the publication of this dissertation, is the maximum of all information sources. After the generation of this auxiliary gridmap is forwarded directly to costmap2D and the output is the local gridmap around the vehicle. The configuration for the grid is variable and for the iCab project scenario, it has been used the resolution of 0.7m and 42x42 cells (30 x 30m) where the vehicle is in the middle of the map. The update frequency of the costmap module is set to 5Hz but the load of the computer allows this update frequency between 1 and 0.5 Hz.

For the global map, it is necessary to create two gridmaps. One for the initial localization AMCL node which takes exactly the map created with the sensors, without any posterior

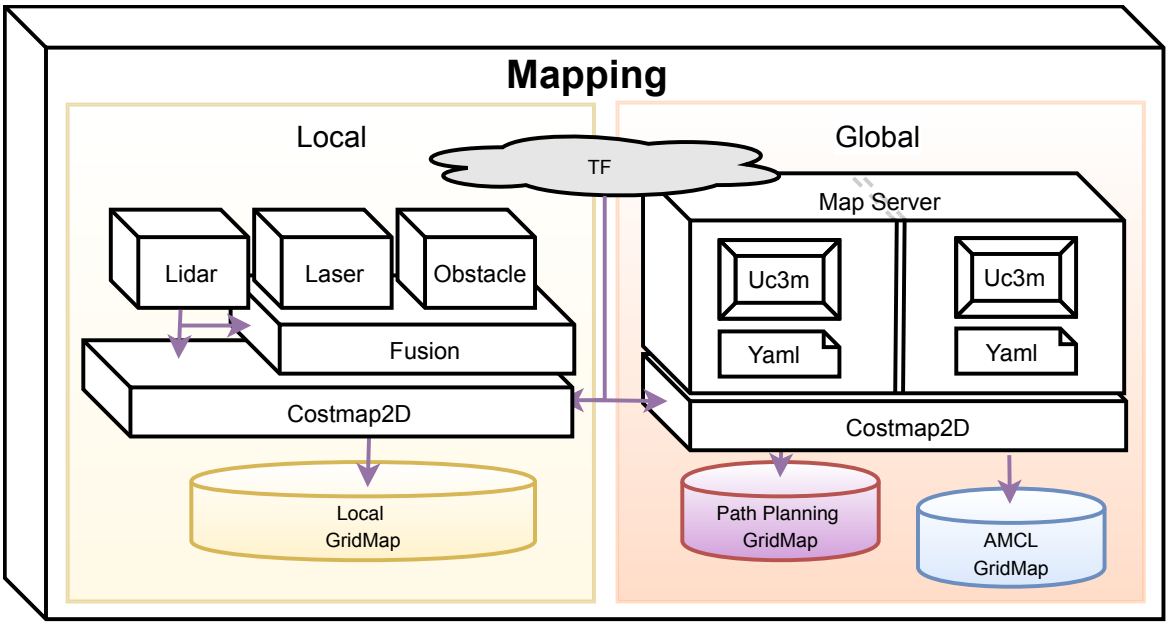


Fig. 5.11 Mapping module architecture workflow.

modification; the other one is for path planning where the global planner determines the best trajectory using the modified global map where there are prohibited zones such as gardens or corridors. The difference between maps are visible in figures 5.12, and 5.13. These global maps are published only once per request, that means, there is no updated until another node requires it in order to save bandwidth.

The inputs of this mapping module are PCL, LaserScan and Obstacle messages from perception; and TF for transformations. The outputs are local Gridmap and two global gridmaps.

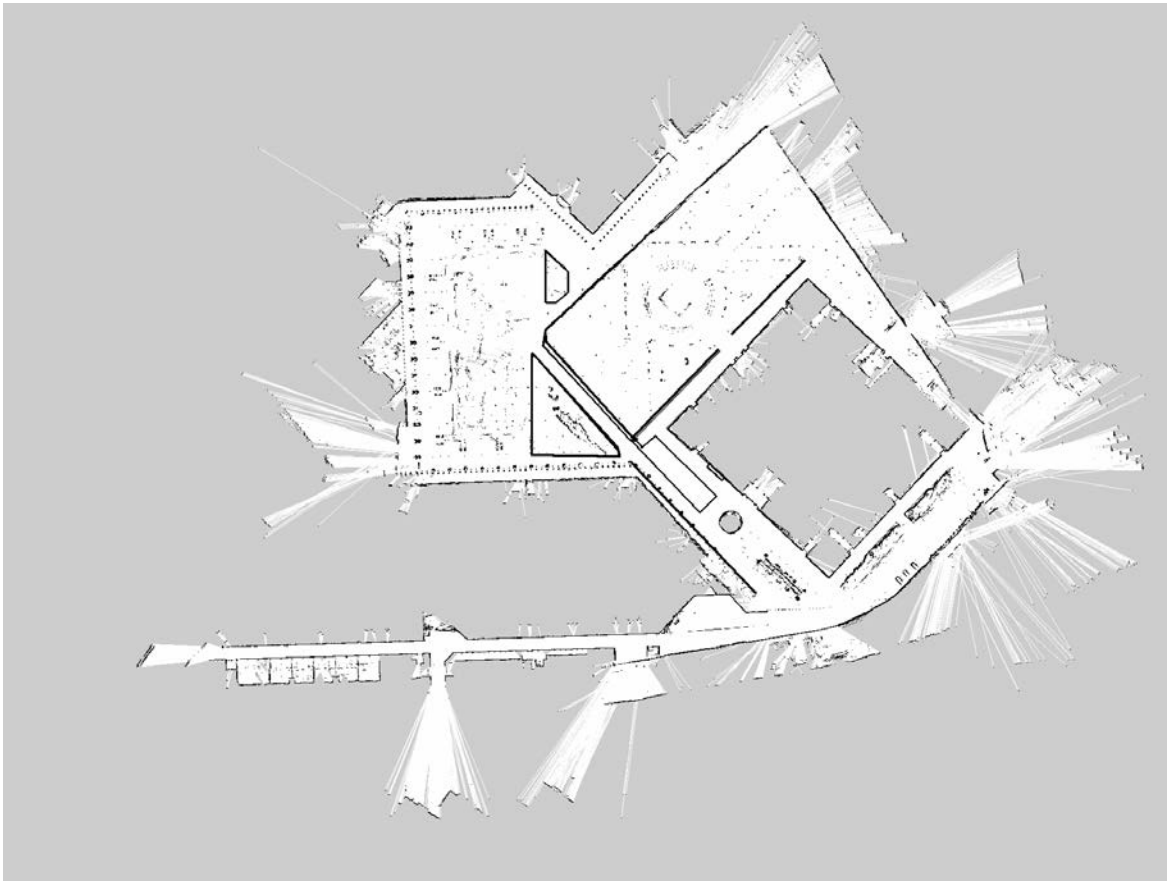


Fig. 5.12 Planning global map.

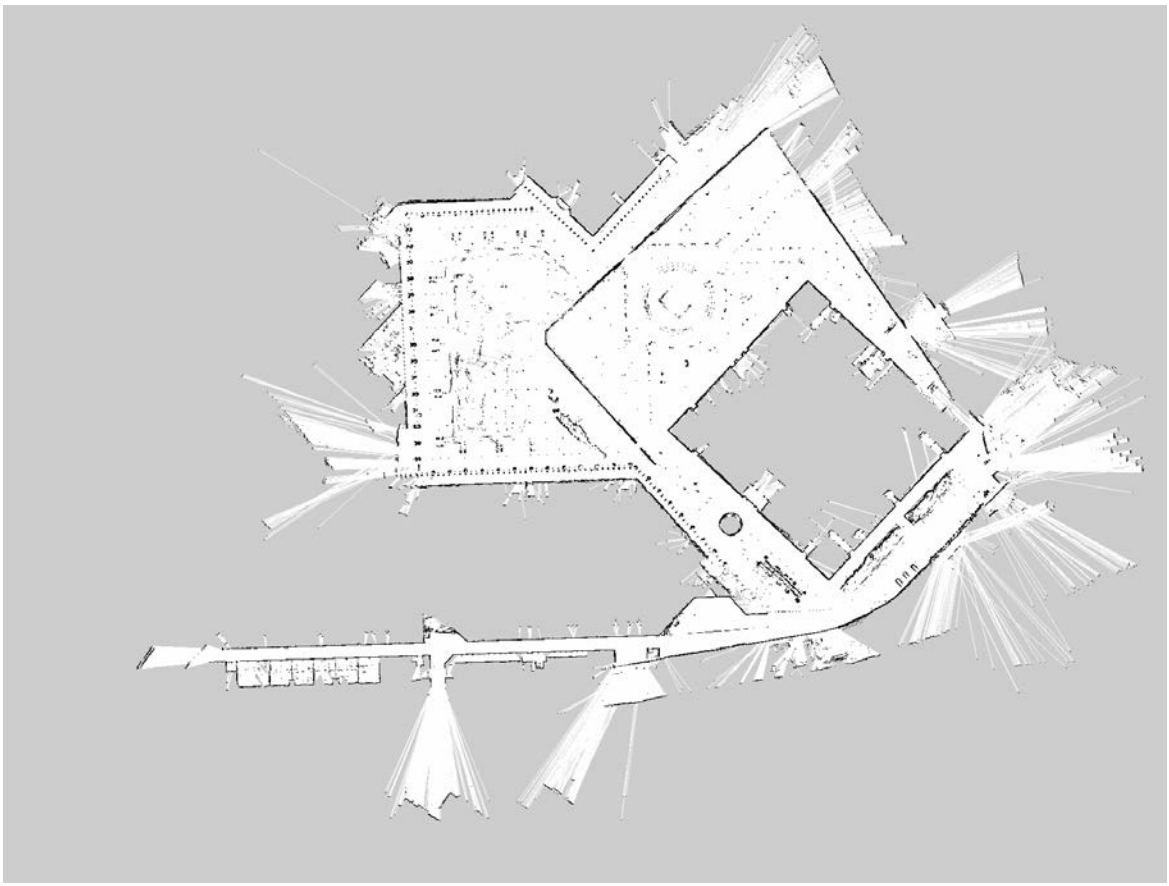


Fig. 5.13 AMCL global map.

5.4.2 Localization workflow

Figure 5.14 shows in detail the structure of the localization module. The aim of this module is to locate in the global map the vehicle and take into account the movement in real life and transpose it to digital. Odometry Manager is responsible for local and global odometry messages and the aim of generating the correct transformations between reference coordinates using TF broadcasters. These transformations are very important for upstream modules in the architecture which will fail if this module is not accurate. The TF cloud in color in figure 5.14 indicates that the TF in green and blue is updated with both local and global transformations; the one in grey, it is not updated with any of the transformations and only needs the \map frame.

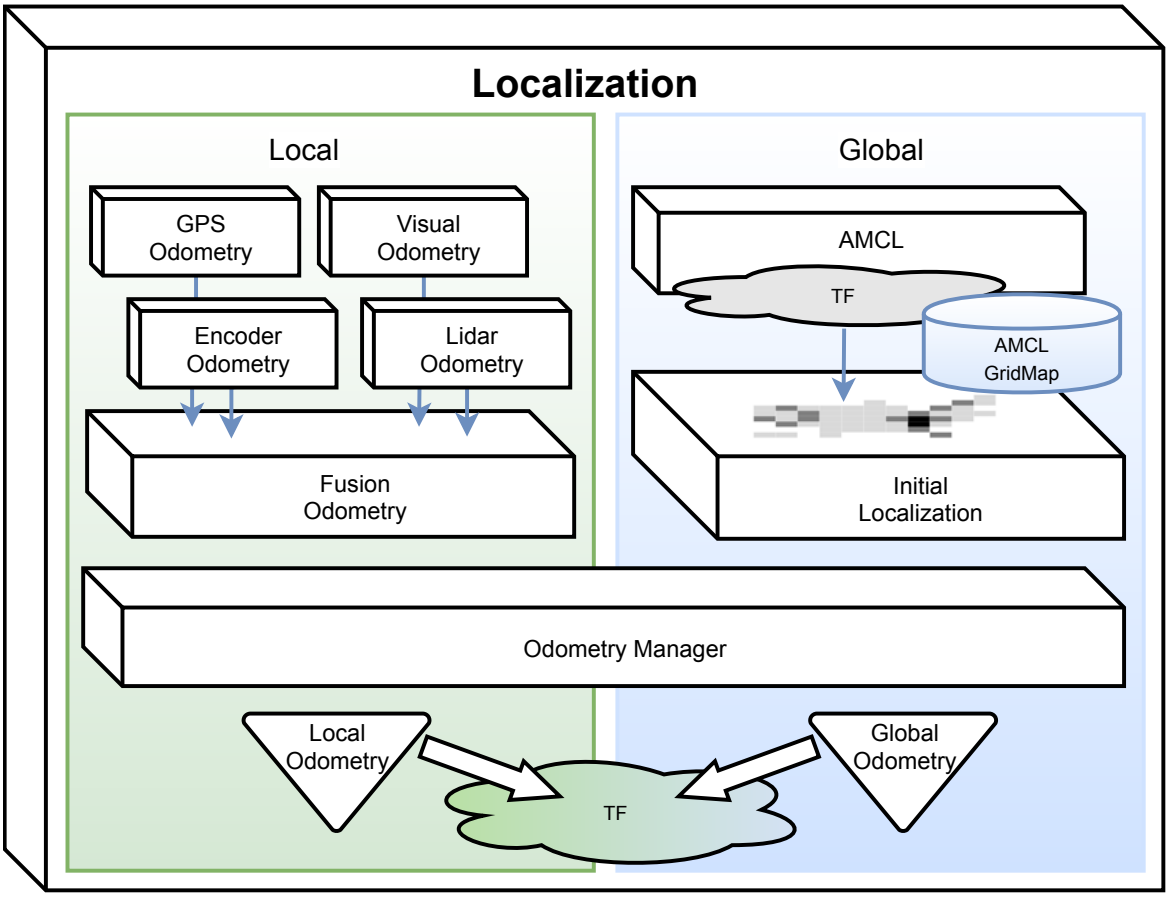


Fig. 5.14 Localization module architecture workflow.

Local odometry message represented as the inverted triangle in figure 5.14, is the outcome of the Odometry Manager which selects the odometry to be used upstream in the architecture. The selected odometry is done by parameter configuration in the configuration file startup process. Additionally, Odometry Manager is responsible to broadcast the transformation

between frame \odom and \rear_wheel_axis using TF library. Figure 5.15 represents the selection of the local odometry between all the possible odometries and the TF generation.

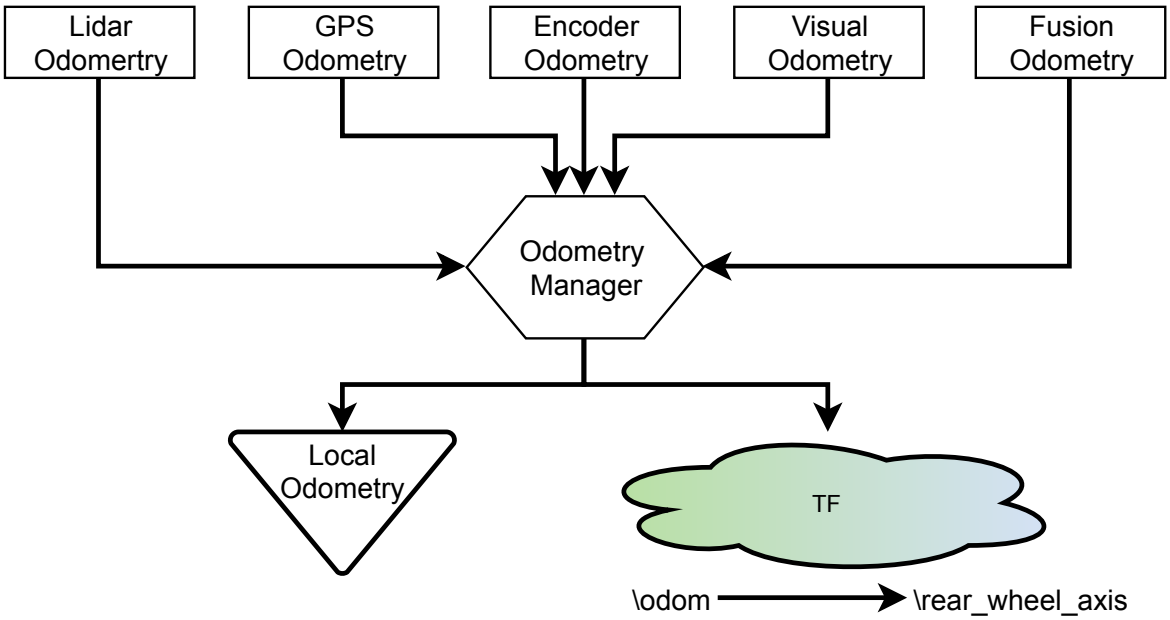


Fig. 5.15 Local Odometry selection.

Global odometry message is forwarded from AMCL which is responsible to generate the initial localization and broadcast the TF between \map and \odom. AMCL needs the sensor data to compare with the global gridmap (AMCL GridMap), local odometry and initial guess. This global odometry is the transformation between \map and \rear_wheel_axis in order to work with global coordinates for upstream use in Planning module.

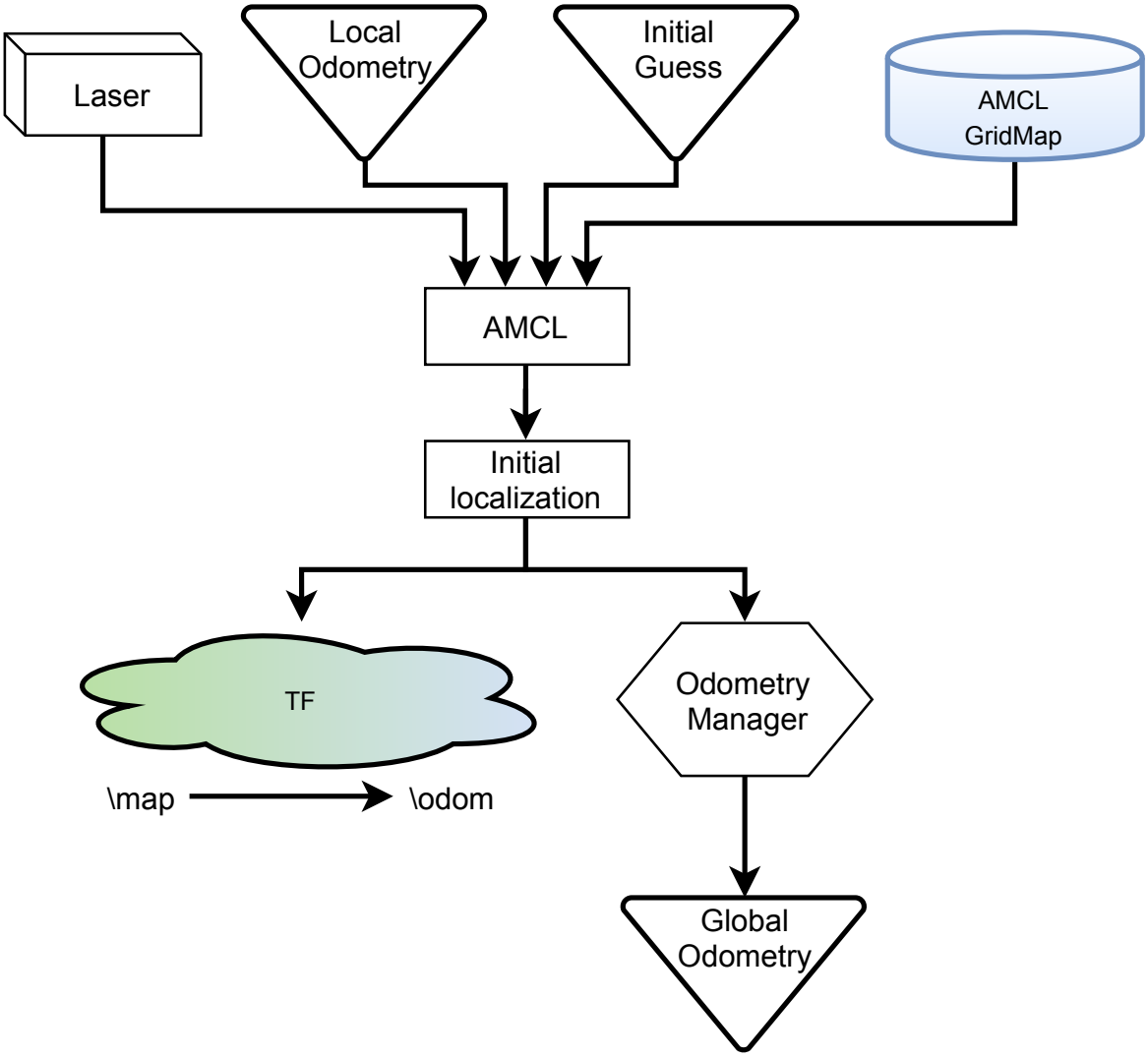


Fig. 5.16 Global Odometry generation and TF.

5.4.3 Path Planning workflow

This module is responsible of the trajectories, global and local, using gridmaps and current localization represented in figure 5.17. This module is special due to the sequence of procedures.

Firstly, a global trajectory is generated using the global gridmap (Path Planning Gridmap) and TF for the current position using the transformation between \map and \rear_wheel_axis. The gridmap used contains the boundaries manually added where the vehicle is not able to navigate. The method used is A* or Dijkstra previously configured, which generates the global trajectory from the vehicle to the goal point. Additionally, it is published the *Path* message, a collection of waypoints, where the vehicle can navigate. After this generation, the local planning uses the global path to achieve the overall trajectory. This distinction between trajectories is due to the ability of the local planner to modify and recalculate the trajectory with dynamic obstacles.

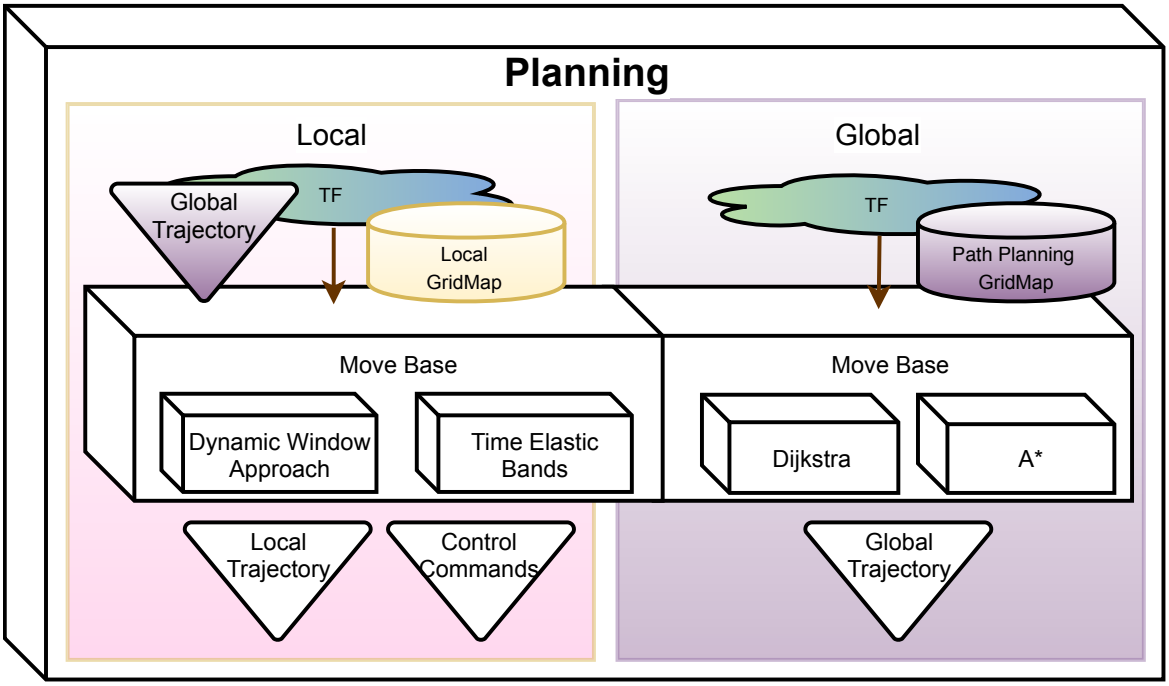


Fig. 5.17 Planning module architecture workflow.

The local path planner tested for this project has been TEB local planner which uses Time Elastic Bands but the architecture enables the use of the Dynamic Window Approach which is not optimal for Ackermann vehicles. In order to create the local trajectory with a configurable lookahead distance, TEB local planner uses previously generated global trajectory. This method is not the optimal approach for Ackermann vehicles due to the avoidance strategy of

dynamic obstacles (figure 4.39 shows the recalculation of local plan in a wrong direction). In work [61] a complete study of obstacle avoidance and a comparison between global and local trajectory is done.

5.4.4 Control workflow

The generation of the *AckermannDriveStamped* messages which contain the desired speed and steering angle is done by two different approaches. The first one is directly the outcome of the TEB local planner which also provides the velocities and steering angle for each waypoint in the local trajectory. This speed and velocity are based on the previous position and the time needed to reach the next waypoint with maximum velocity and steering angle limitations. The second approach is displayed in figure 5.18 which takes the local plan and compare it with the current position. This method is called Optimal Path Controller and uses a Linear Quadratic Regulator (LQR) in order to keep the vehicle in the path given. This module checks the behavior selection of the vehicle and overrides the commands generated from the regulated control stage.

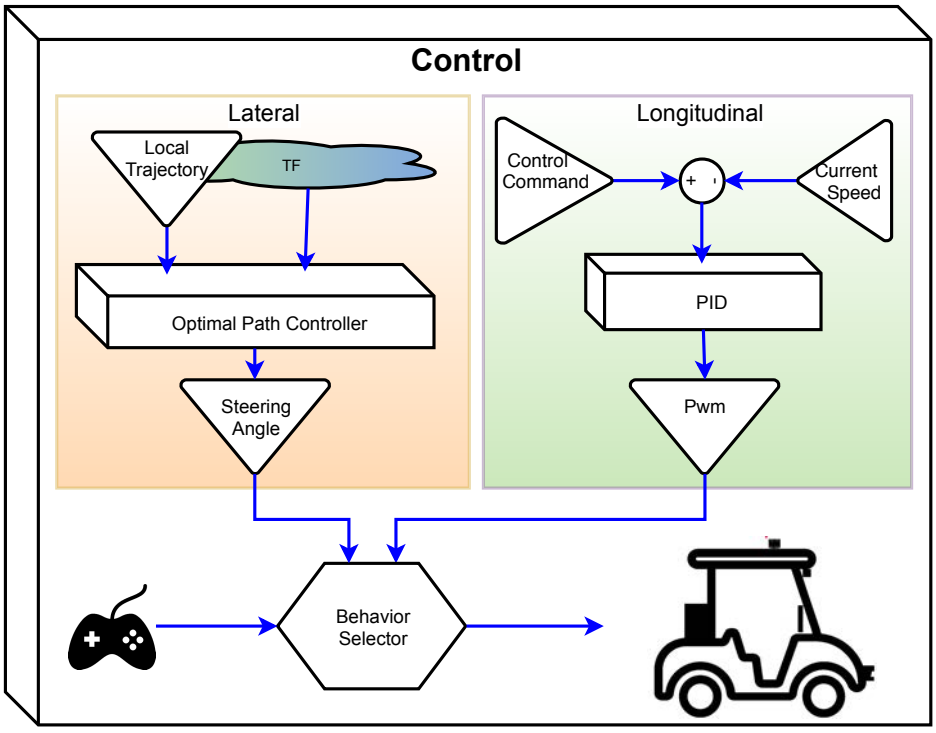


Fig. 5.18 Control module architecture workflow.

5.4.5 Behavior workflow

This module is in charge to select the behavior of the vehicle. Figure 4.41 shows all possibilities of the state machine. Each task is defined by a flowchart in figures 5.21, 5.22, 5.24 and 5.25.

One important aspect related only to the iCab vehicle and not to the architecture is the node *movement_manager* which acts as the bridge between ROS and PIC micro-controller of the steering angle and traction motor. This node is special due to the ability to accept commands from topics and from services. The joystick commands in manual behavior are managed as services and the commands generated in the control module are managed by topics. This distinction is critical in order to receive the acknowledge from the server, in the case of joystick commands, because it is necessary to check the health of the order and the vehicle should warn if the command will be processed or not. In the case of control, the closed loop of velocity and steering will check directly if the order is accomplished or not and adjust the values sent to the micro-controller.

The node responsible for the behavior selection, figure 5.19, is at the GUI *icab_reconfigure* which name only indicates the possibility to reconfigure aspects of the vehicle and project. The user can manually change the behavior with the buttons associated in figure 4.43. Each task is described in the node *reactive_tasks* which contains the logic associated to accomplish each task. The name of the node is only provisional and in future work will be changed.

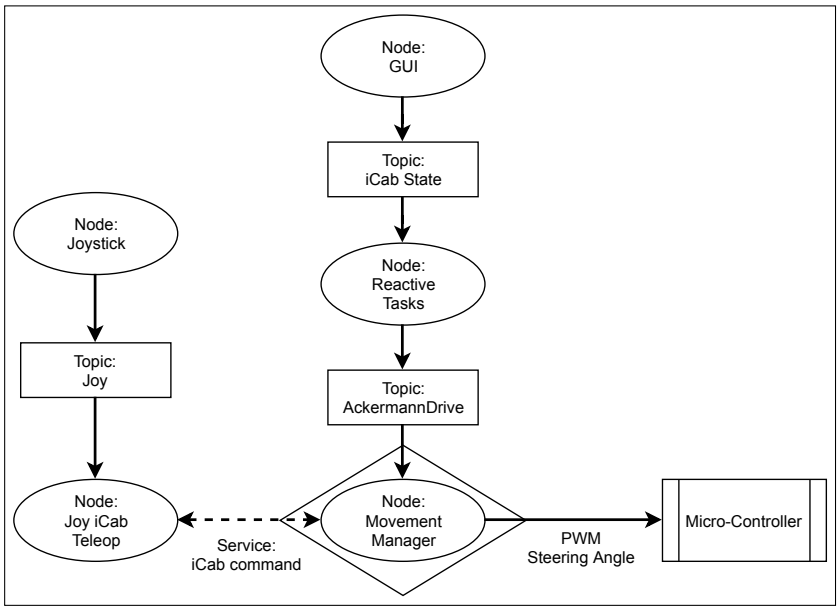


Fig. 5.19 Behavior selection from GUI.

Manual mode

This state machine overrides the commands generated from the control stage. It is the only one that uses services to create the movement of the vehicle. The commands obtained by the joystick driver generates PWM rotor and steering angle, depending on the button pressed, and this command enters directly to driver module called *movement_manager*. This module is the final responsible to send the commands to the PIC micro-controller in form of *TransInfo* and *DirInfo* message. The flowchart of manual behavior is displayed in figure 5.21. Notice that movement manager acts accordingly to the state machine when manual mode is pressed. The joystick configuration buttons have been developed in time for the best and friendly control of the vehicle and finally has been decided the setup displayed in figure 5.20. It is important the decision to control the vehicle in acceleration and not in the speed because the feeling of control is better when the user can accelerate or brake like in the Ackermann vehicles.

- Acceleration: It is controlled by adding or subtracting PWM to an accumulated value. One button of the joystick add +5 units to the accumulated PWM and another value subtracts -5 PWM units. In the beginning, when the manual button in GUI is pressed, the accumulated acceleration is 0 until a button is pressed. Two different buttons reduce the accumulated value to 0 in the forward direction and two of them reduce the accumulated PWM to 0 in the backward direction. This distinction is due to the stator polarity.

- Steering: It is controlled by a variable which tracks the steering angle adding ± 1 with the 8-way D-pad and ± 5 with two buttons.

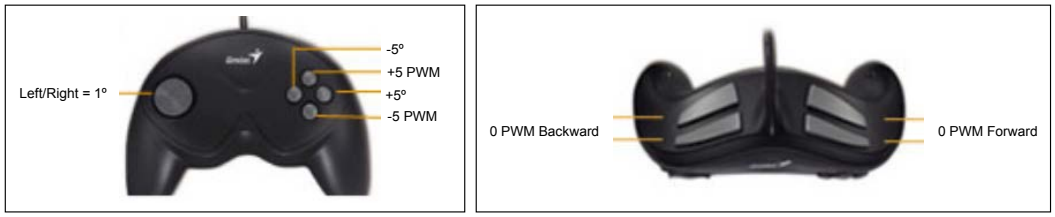


Fig. 5.20 Joystick layout.

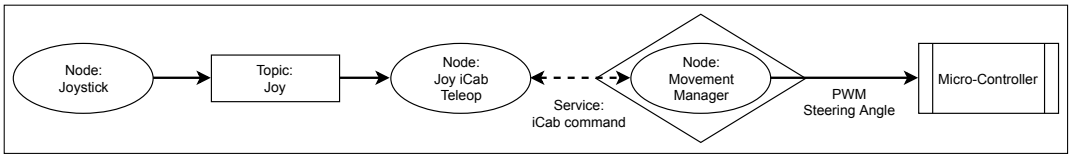


Fig. 5.21 Manual behavior.

Follow Left/Right and Between Boundaries mode

The aim of this three tasks is to keep a distance previously defined between the vehicle and/or the left or right boundary. The boundary is calculated using the frontal monoplane laser. This distance is used to calculate the steering angle with a simple proportional error value. Figure 5.22 displays the workflow and the steering angle generation for these tasks. The speed is selected by a parameter which is possible to configure at the beginning of each task. Figures 5.23 and 5.24 shows the workflow of the follow left, right and between boundaries.

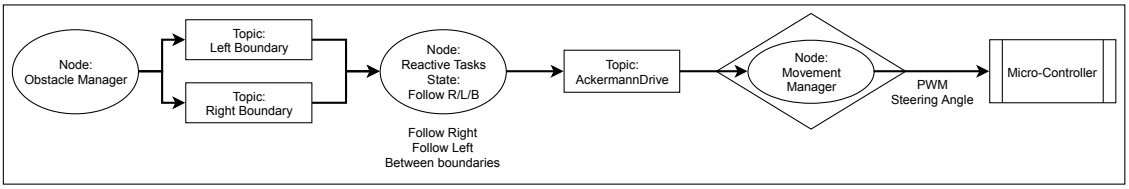


Fig. 5.22 Obstacle manager to movement manager workflow.

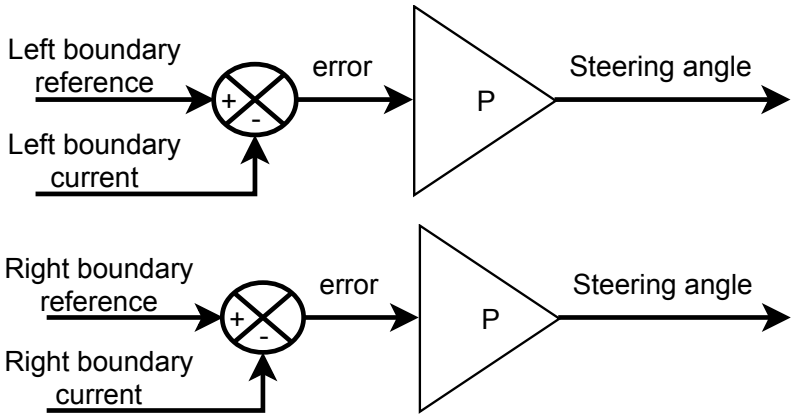


Fig. 5.23 Description and workflow of the Follow left/right boundary tasks.

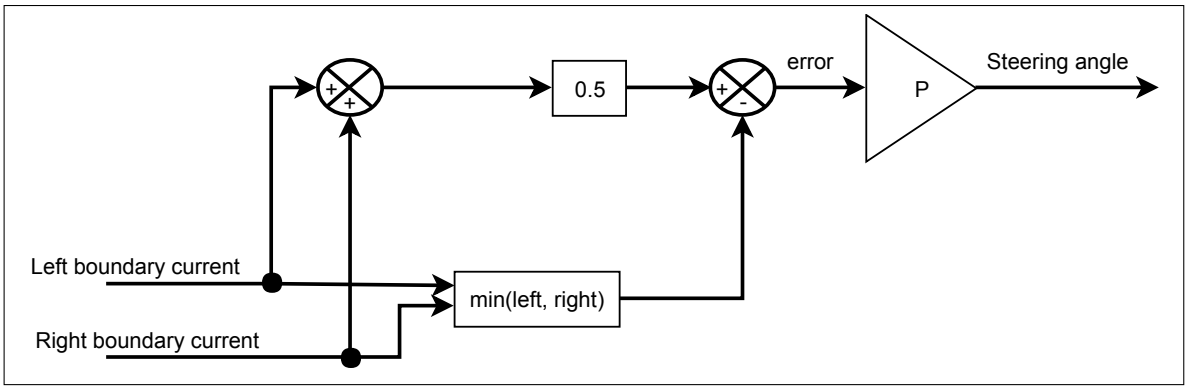


Fig. 5.24 Between boundaries task mode workflow.

Vehicle follower mode

This task is considered as a cooperation task because of the shared information between vehicles and the performance. The work in [40] describes the methodology and experiments done in iCab vehicles for testing the cooperation and communication scheme. It highlights the importance of Vehicle-To-Vehicle (V2V) communication and shows the advantages of Vehicle-To-Pedestrian (V2P) and Pedestrian-To-Vehicle (P2V) communications when added to the system environment perception. Several experiments have been performed for real-life scenarios to verify the results, and evaluate the performance of the proposed approach. The experiments outcomes prove the high performance of the proposed approach in real-world application and emphasize the importance of V2X communications in autonomous vehicles. Figure 5.25 describes the sensors used to keep the distance between the leader and the follower with shared information over V2V via VPN.

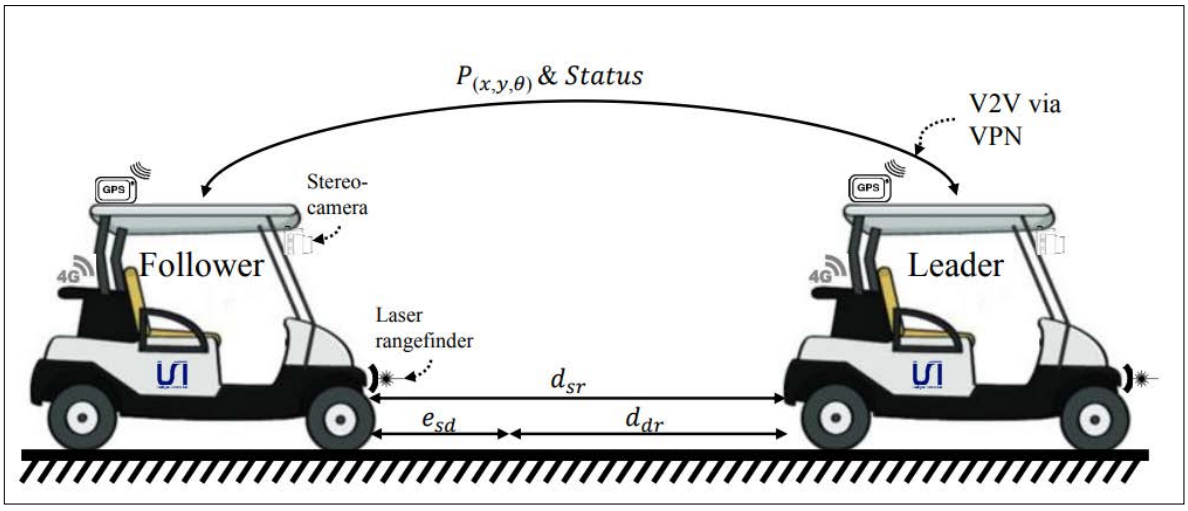


Fig. 5.25 Vehicle follower behavior.

Autonomous task mode

The whole iCab project is defined in this task, where the vehicle is able to navigate around the campus using localization, maps, and planners in order to generate control commands. The description of this task is in chapter 6: iCab use case and results.

5.5 Architecture Layers

After the explanation of the modules involved in the architecture, there is one important element in the workflow of the architecture to discuss. The modules describe only the behavior and the performance but in an abstract concept, the architecture is divided into three different layers in a scale from high deliberative performance to reactive layer. Figure 5.26 shows these three layers where the top layer is the deliberative layer based the decisions on maps and planners, the middle layer mixes task with components reactive and deliberative and the low-level layer use the sensors to react to specific events. The work presented in [62] explains in detail each layer and performance in two scenarios.

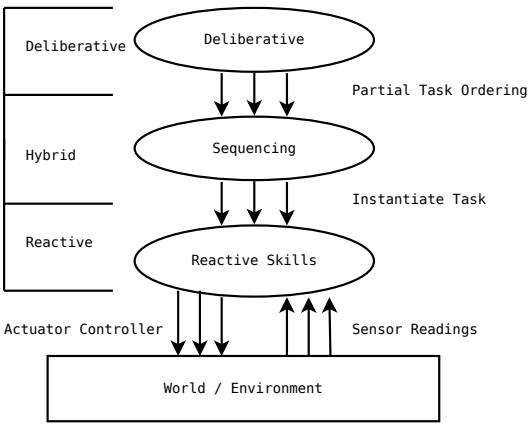


Fig. 5.26 Three layers of the architecture.

The reactive layer is a component in *Movement Manager* node which enables or disables the movement of the vehicle. This component is based on the message *ObstacleThreat* explained in figure 5.10 and 4.42 where it tracks obstacles in front of vehicle. If there is any obstacle dynamic or static inside of this safety distance, *Obstacle Manager* publish the threat and *Reactive Tasks* will act accordingly, at the moment, stopping the car. Due to the evolution of the project, the architecture considers different alternatives when an obstacle is inside of the safety distance and instead of stopping the vehicle, activate an avoidance performance.

Chapter 6

iCab use case and Results

The iCab project is divided into navigation, multi-task allocation (which includes cooperation) and communication. Due to the dimension of each field, the results section of this dissertation is focused only on the navigation part which is directly linked to the navigation architecture of the vehicle. The task allocation and cooperation are explained in detail in Control and Communication Systems for Automated Vehicles Cooperation and Coordination [37]. Multi-task allocation needs communication between vehicles, infrastructure, and pedestrians which is explained in the publication named V2X communications architecture for off-road autonomous vehicles [48]. For the Human Machine Interface of the iCab project, a web server has been developed based on the work of the final project called *Modelo de conocimiento para la homologación e inspección de servicios de transporte basados en el uso de vehículos autónomos* [56].

The main objective of the experiments proposed is to demonstrate that the architecture is functional and the requirements proposed at the beginning of this dissertation are accomplished. In order to demonstrate that, the CPU load must be analyzed meanwhile the vehicle is in autonomous mode. In addition, it has been analyzed the steering angle and speed for all the scenarios. There have been proposed two operating modes for the main CPU: one where the navigation modules such as AMCL, global and local planner, costmap2D, GUI, and controller are running without duplicate information for localization and mapping. The other experiment is intended to analyze the performance of the CPU usage for the same nodes when there is an overload in the CPU usage due to redundancy information for localization and mapping. This redundancy information is generated in order to improve the readings with fusion and test the maps generated standalone without costmap2D. The integration of this duplicate information is not integrated yet but is a task in future work for improvement.

The minimal configuration for the navigation purposes is configured by active modules such as AMCL, lidar odometry, TEB local planner, move_base and costmap2D in addi-

Table 6.1 Time required for Costmap2D using different sources.

Source	**Time [S]**
Laser	0.4
Lidar	>5
Fake laser	∼4.5
Fusion local standalone map	>10

tion to the GUI, movement_manager and behavior selector. For the research configuration and overload, modules which involve the generation of different odometries such as visual odometry, encoder odometry, and GPS odometry are running in parallel without taking into consideration for navigation decisions because the best outcome is the lidar odometry. Other modules such as obstacle detection and classification in addition with the standalone maps for each sensor and the fusion of all of them are not integrated with the costmap2D used for navigation due to the poor update rate of shared messages and the huge amount of CPU load required (table 6.1). In future work section is described the possible improvements for costmap2D update rates which is the main bottleneck of the whole architecture, without updated information of the environment, local trajectory planner is delayed and the performance is poor.

In summary, there is only one functional navigation configuration (figure 6.1), which involves:

- **move base** which is in charge of checking if the goal has been reached and if the navigation commands are generated at the specified rate with updated information of the costmap2D.
- **AMCL** provides the transformation from $\backslash map$ and $\backslash odom$ which affects to the global localization of the vehicle.
- **lidar odometry** is used for the local odometry and provides the transformation between $\backslash odom$ and vehicle reference $\backslash rear\_wheel\_axis$. *Dijkstra* is the global planner used in the global costmap2D. *TEB local planner* is the local planner based in Time Elastic Bands which provides a useful trajectory and speed and steering control commands in order to follow the provided trajectory. *GUI* is the interface for the user to control the behavior selector of the vehicle. *Obstacle threat* is the reactive emergency stop based on the frontal laser.

For testing the CPU usage in the overload configuration (figure 6.2), the extra modules running in parallel are:

- **PCL map**, **Laser map**, **Obstacle map**, **Fusion local map**, are modules in charge of generating the local maps based only in the sensor data. For PCL map, the sensor used is the lidar 360 degrees, for laser map is based in the frontal laser 180 degrees, for obstacle map is based on the stereo camera and the fusion local map is the integration of all maps.
- **Encoder odometry**, **Visual odometry** and **GPS odometry** generate the local odometry.
- **Bird view**, **Free space mask**, **Obstacle mask**, **Calibrate extrinsics** are in charge of the image processing in order to acquire the obstacles map.

In addition, in order to test and compare for research purposes, there is another configuration where the vehicle is able to run and administrate the modules for the functional navigation in addition to standalone maps, fusion map, duplicated and fused odometries and stereo image processing (figure 6.2).

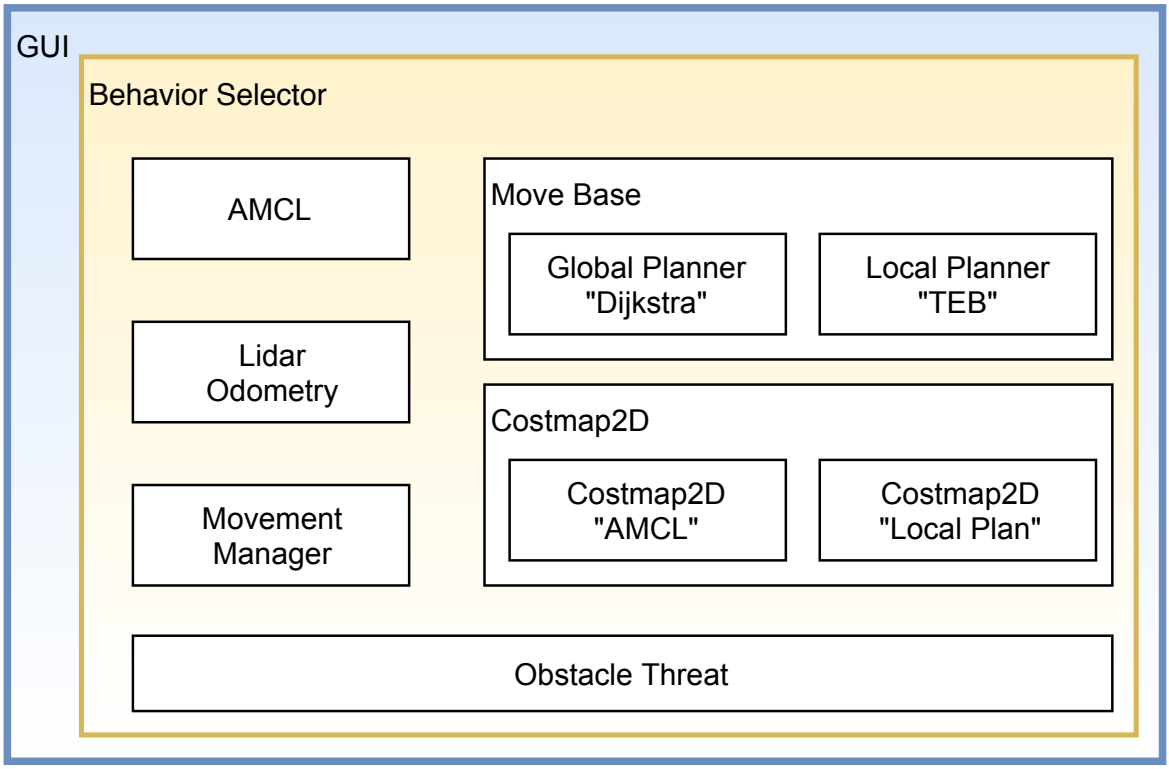


Fig. 6.1 Minimal configuration for navigation purposes.

In order to measure the performance of the vehicle, it is assumed the reception of goal points located in the University Carlos III de Madrid, Leganés Polytechnic campus. It also

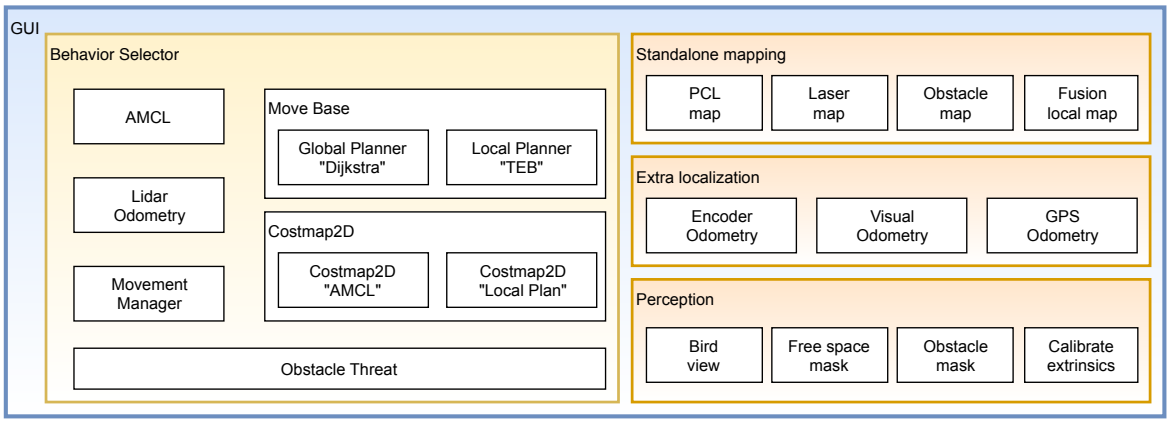


Fig. 6.2 Maximal configuration for research purposes.

assumes that the guess of the initial localization is provided by the GPS. It is a guess due to the fact that AMCL needs an initial region to spread the particle filter. Another assumption is the weight of the vehicle which is constant and at the moment of the tests were only one passenger inside of the vehicle in all the experiments. The final assumption is the activation of the movement of the vehicle using the GUI and the touchscreen. That means the vehicle does not move even when there are trajectory generation and control commands generated from the planners until a button in the GUI is pressed. This button is critical at this stage of the vehicle to understand that the passengers are already inside and ready to start the journey.

6.1 iCab use case

The tests are divided into two scenarios where the experiments took place with different configurations. In order to test the efficiency and performance of the modules involved, a CPU study has been done, taking into account the CPU load along the experiment for each process working in the vehicle. One indicative of a healthy system is the computational load of its CPUs which will be able or not to accomplish the delivery of the messages in time.

In all cases, the experiment starts the same way (see figure 6.3), the vehicle is located at one pick up point waiting to receive the goal point. The reception of the goal point is done manually as previously said but the architecture accepts this goal point as a message coming from the multitask allocator. The next stage is the generation of the global path in a map of 1696x1280 (508,8 x 384m) using the Dijkstra method and the global costmap2D. Next step is the generation of the local trajectory with the control commands for steering and velocity provided by TEB local planner. The vehicle waits until a continue button is pressed in the Graphic User Interface. In order to keep the vehicle in the place, the brake system is active

while the vehicle is in pause mode. When the button is pressed, the vehicle starts moving at the desired speed provided by the TEB local planner and following the local trajectory which its *lookahead distance* is a waypoint located in the global trajectory at 8 m. The costmap generation is done by monoplane laser located in front of the vehicle and not with the lidar 360 due to the limitation of readings near the vehicle caused by the physical location at the top of the vehicle which is not able to detect obstacles near to the vehicle. Another drawback of using the lidar for the costmap2D generation is the time required to process the point cloud which delays the local map delivery up to 10 seconds each, which makes impossible the use this sensor. Using a technique called *fake_laser* which takes the point cloud of the lidar and extract the points in one plane at specific height reduce the points considerably but still the processing time for 360 points (one beam per degree) at a range of 60m, the output local costmap2D is generated every 5 seconds, still impossible to use with dynamic obstacles and local planning generation. The use of the frontal laser allows the local costmap generation average of 0.25 seconds. The control loop for local trajectory is limited to 1 second of old data, which means that will not generate local trajectories if the map is not updated. When the goal point is reached, the vehicle automatically stops and activates the brake allowing passengers to get off the vehicle safely.

The recorded data files are generated with ROS tool called *rosbag* which saves the selected topics in order to reproduce and analyze. This tool consumes part of the bandwidth due to the access and write of the messages in Hard Disk and also consumes part of the CPU. The impact on the performance is not high but it is necessary to take into consideration for the minimal time delay in the average rate of the messages analyzed.

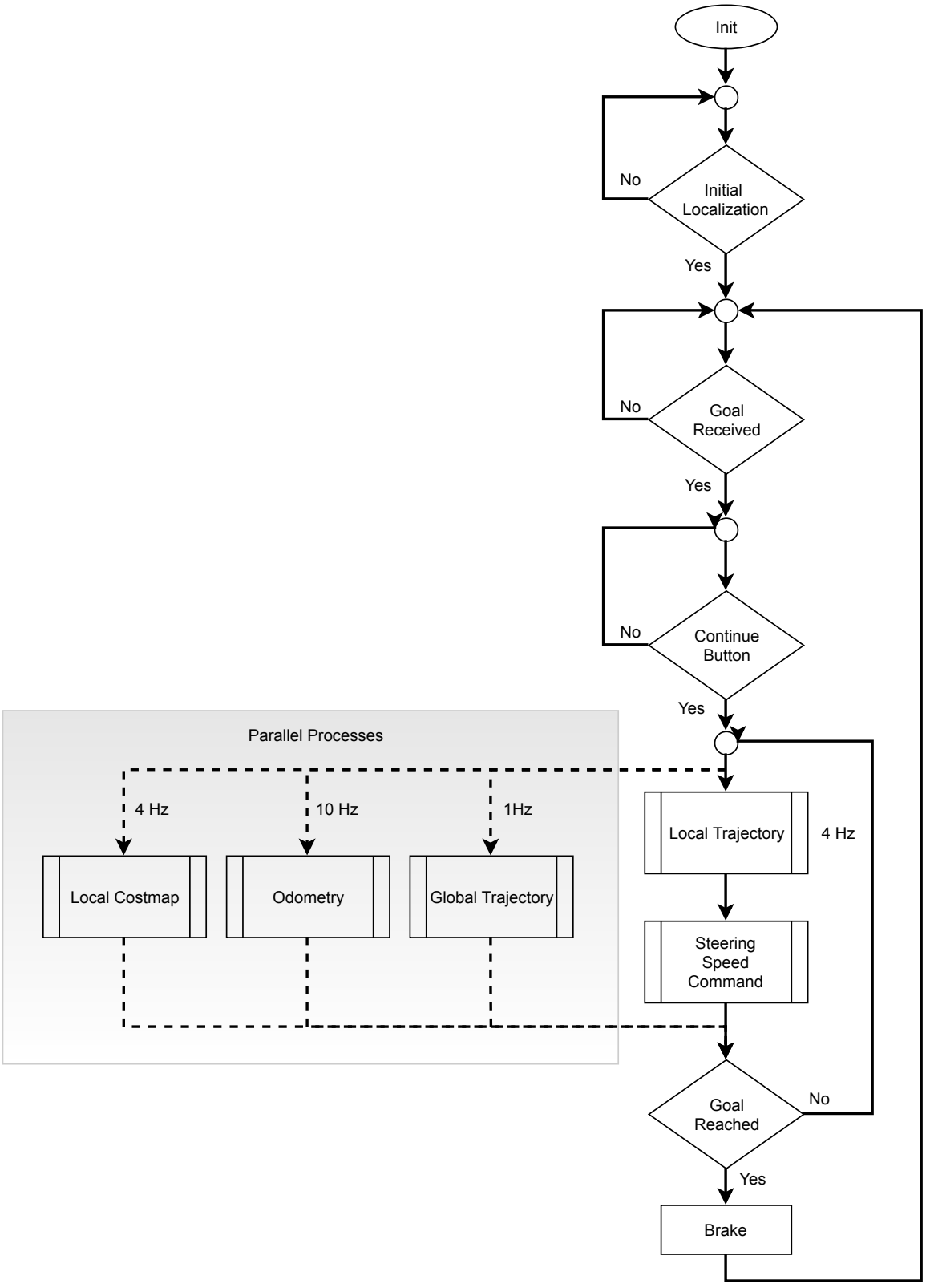


Fig. 6.3 Flowchart for the use case experiments.

6.2 NUC required

During the development stage of localization and odometry generation from different sources, the main CPU revealed a major problem due to the limitation of computational power. In order to know what was happening with the lidar odometry working in the same computer as the planner and mapping, an extra test was done. The test took place in the Sabatini building, where the vehicle navigates four laps clockwise and finish at the same initial spot. Figure 6.4 shows the initial setup in red arrow and the path in a blue arrow. Notice that the corners are softer than in the picture.

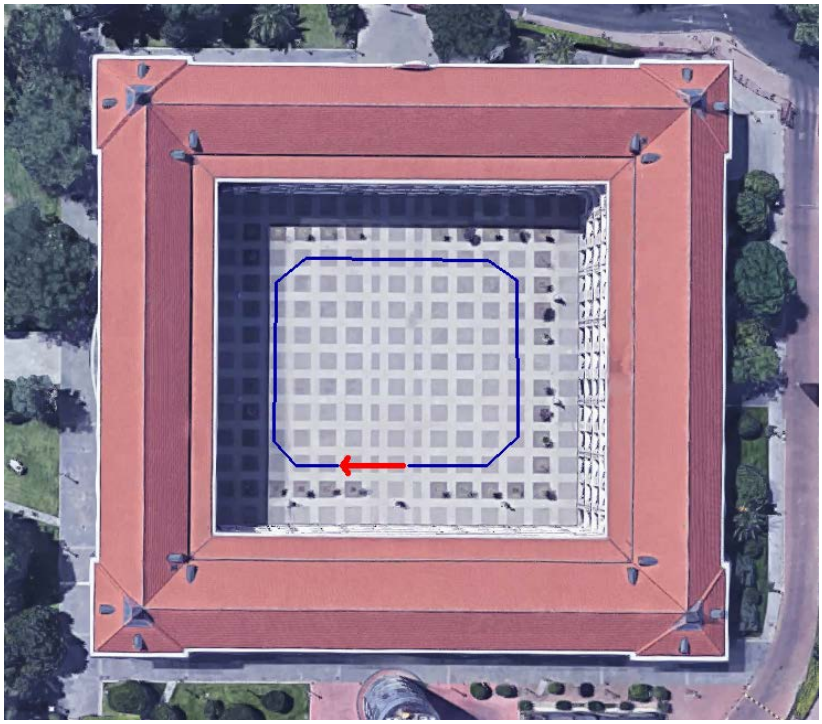


Fig. 6.4 Sabatini zone for testing the odomertry performance without NUC and with NUC.

Figure 6.5 shows the overall rate of publishing of lidar odometry (top left) and the movement of the vehicle (top right) working in the same computer. The total amount of messages received for this experiment was less than 700. The overload of the computer caused the lidar odometry unable to process the PCL points in time publishing poor odometry at 3Hz. For this reason, it was mandatory to integrate a new auxiliary CPU (in this case the NUC) to do the PCL lidar processing in order to gather better odometry at a higher rate of publishing. Figure 6.5 bottom left shows the same experiment with the NUC generating the lidar odometry while the main computer was in charge of the rest of the nodes.

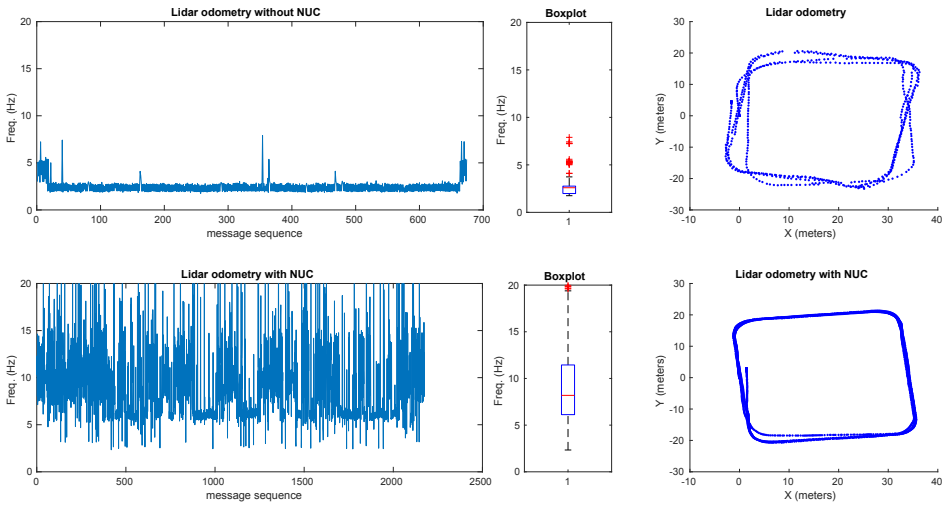


Fig. 6.5 Lidar publish rate and odometry performance in the Sabatini building with and without NUC.

The results are evident when the NUC is running the lidar odometry, the average rate and the performance are improved, maintaining an average published rate between 8 and 10 Hz. The performance when there is no NUC installed in the vehicle is not accurate and the published rate of the lidar odometry messages are between 3 and 4Hz. The necessity of another computer to run the lidar odometry has been proved.

6.3 Obstacle avoidance

In this section, an experiment where set up where the vehicle is required to perform a trajectory with a circular building between the starting point and the final point.Two turns are needed and the whole perform is shown in figures from 6.6 to 6.9. With this test, is intended to check the global plan, local plan, and its recalculation. Additionally, it is possible to appreciate the AMCL repositioning in orientation by looking at the lidar points(colorized particles in the images) and the building's walls. In the beginning, the match between the real sensor and the map is not perfect but the AMCL particle filter relocates the initial point improving the global position of the vehicle. After some odometry and sensor data acquired, the initial position and global position becomes better. In the sequence of figures, it possible to appreciate the global trajectory generated in blue which is not perfect due to the nonderivable zones in the middle caused by the grid map resolution. Instead, the TEB local planner is able to generate trajectories affordable by the vehicle. Despite in some events, the vehicle is not able to follow with accuracy the local trajectory, the recalculation generates new local paths more suitable for the vehicle from the new current position.

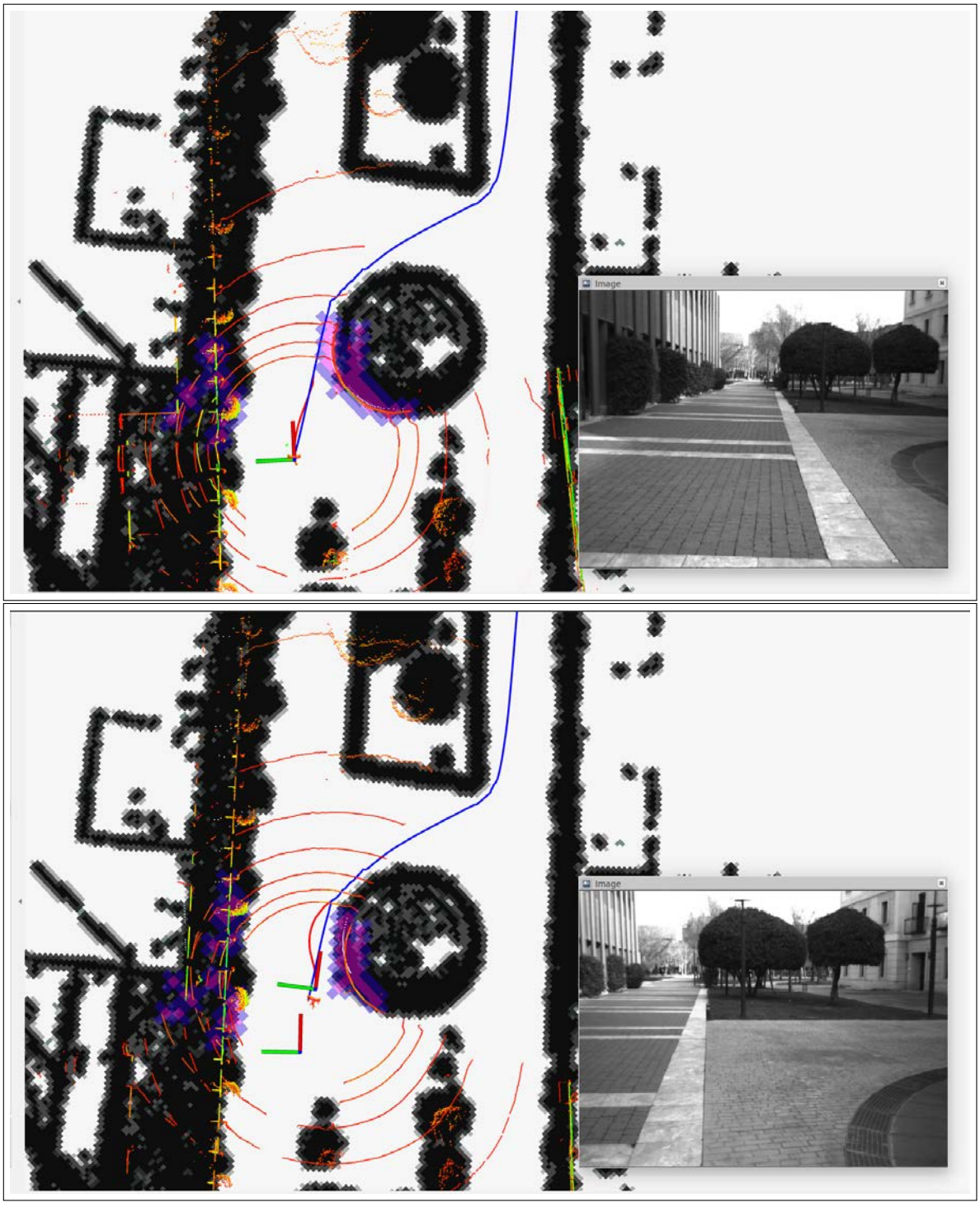


Fig. 6.6 Sequence obstacle avoidance with global and local trajectory generation I.

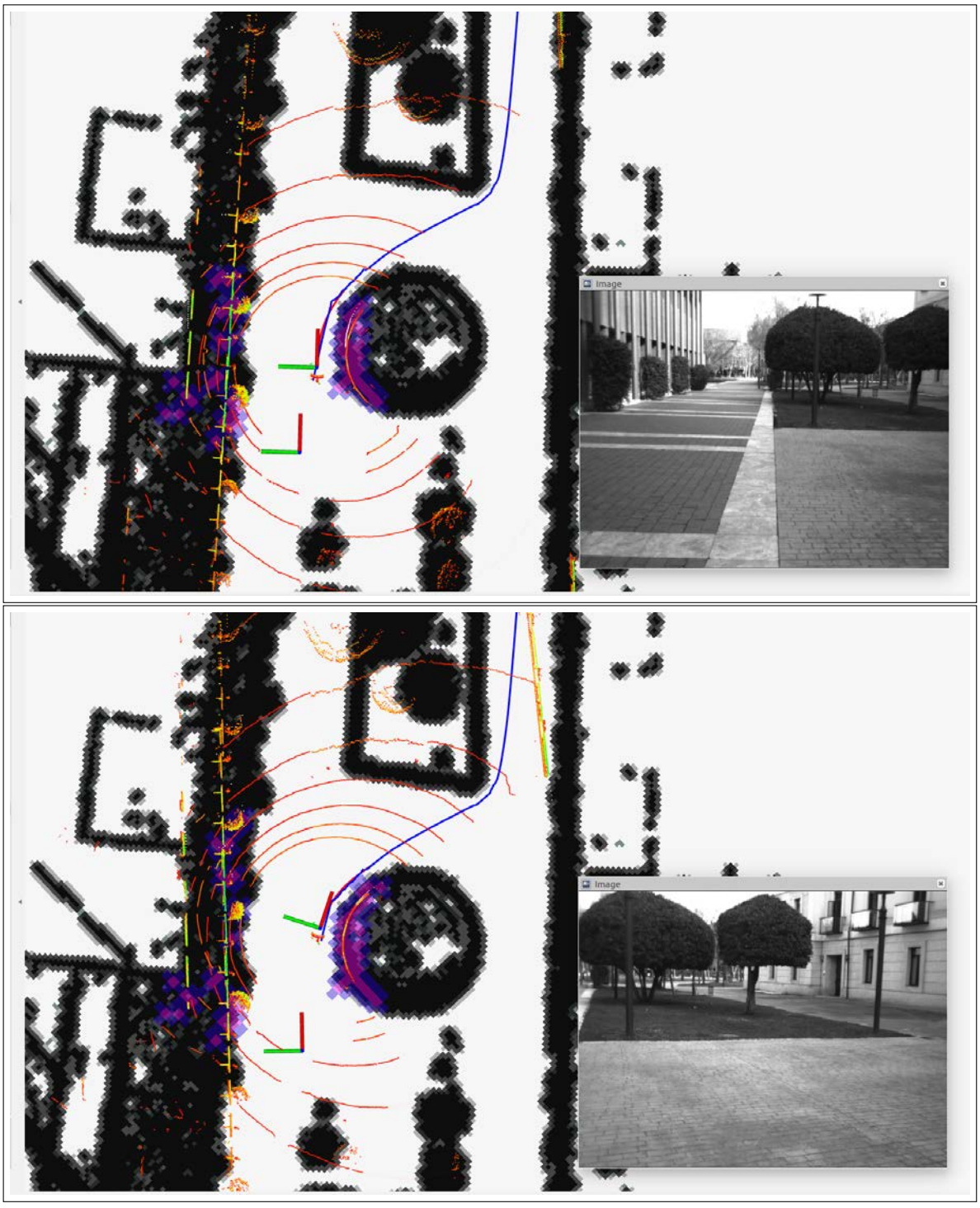


Fig. 6.7 Sequence obstacle avoidance with global and local trajectory generation II.

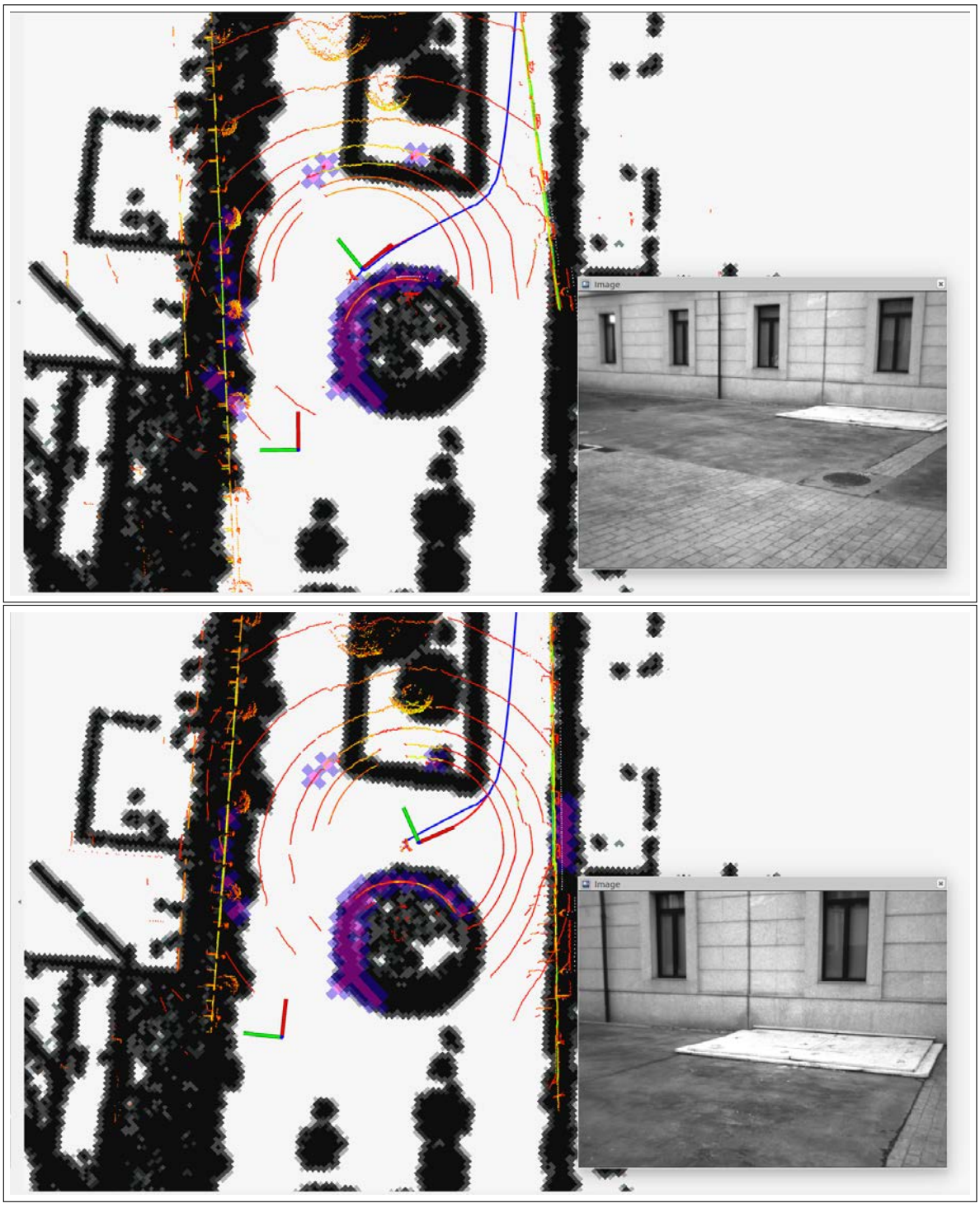


Fig. 6.8 Sequence obstacle avoidance with global and local trajectory generation III.

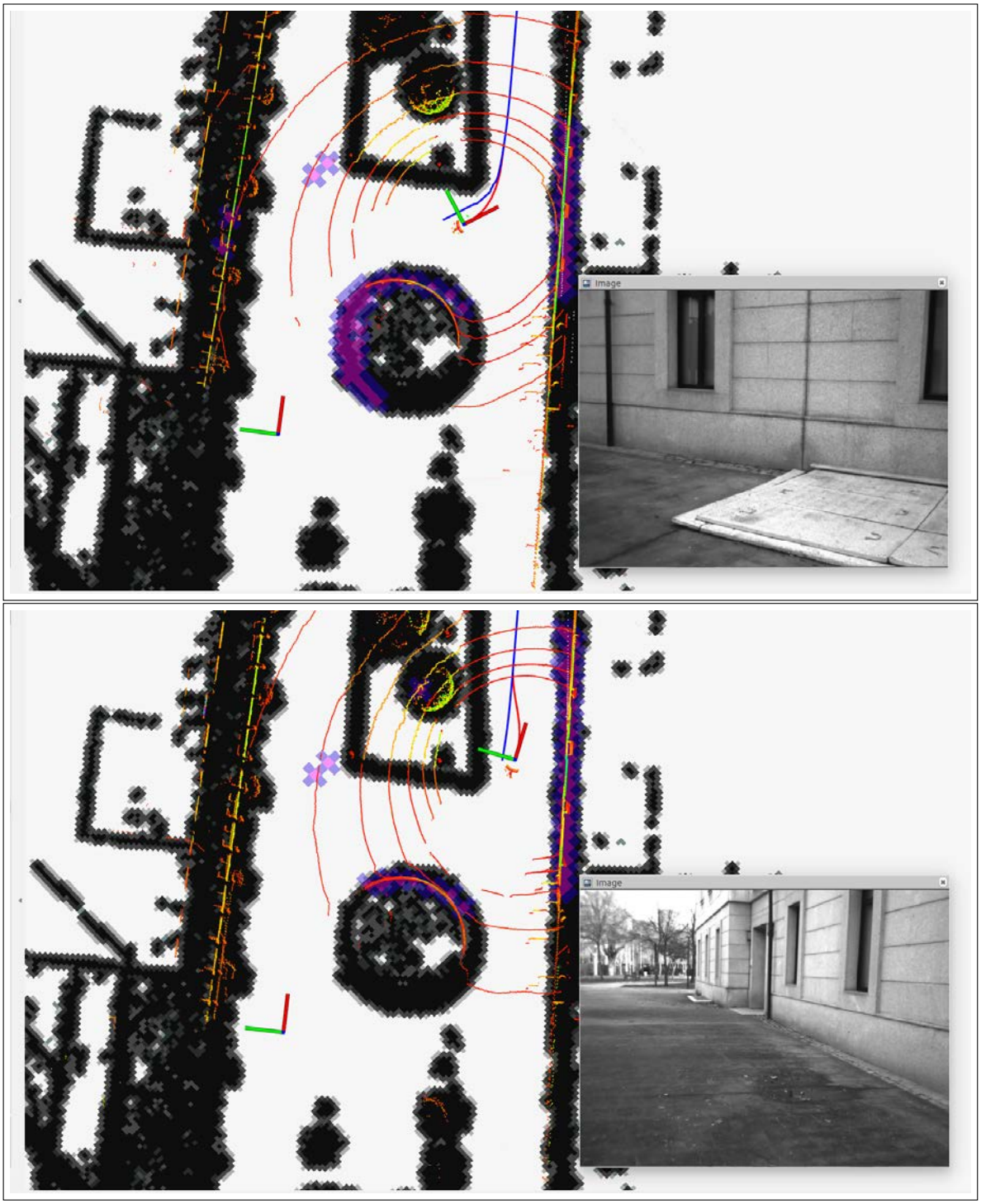


Fig. 6.9 Sequence obstacle avoidance with global and local trajectory generation IV.

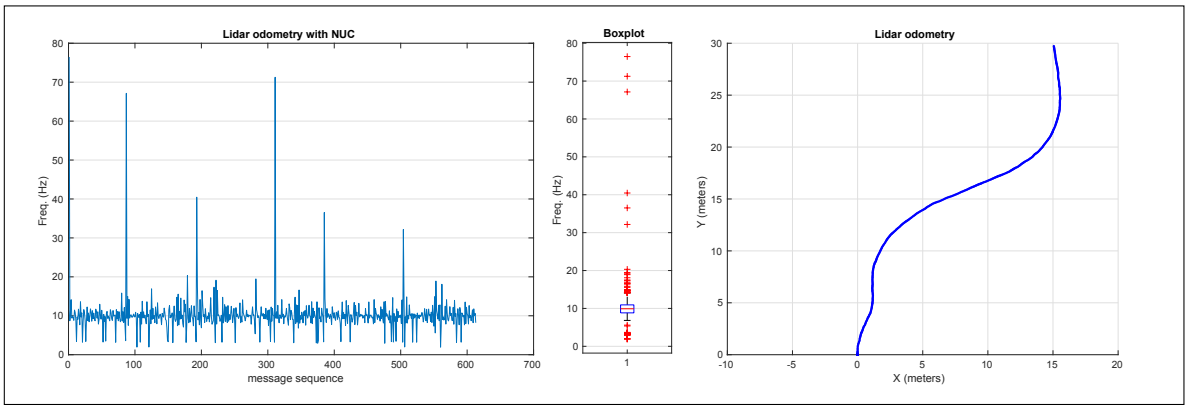


Fig. 6.10 Lidar odometry publish rate and performance.

Figure 6.10 shows the performance in terms of publishing rate of lidar odometry on the left, the box plot showing the average and standard deviation in the middle and the movement of the vehicle on the right. As it can be seen, the odometry performance is excellent in terms of publishing rate compared with the previous experiment where barely reach 4 Hz when there is no NUC computer.

6.4 Metrics, experimental setup and results

For testing the quality of the navigation architecture and performance, it has been measured the time lapse between published messages of the nodes, in particular, the critical messages of the local trajectories, local costmap2D, vehicle odometry and AMCL pose recalculation. Along with the time lapse for messages aforementioned, for each experiment has been quantitatively analyzed the overall trajectory, reference velocity, current velocity, PWM for the linear motor, steering angle reference and current steering angle. Additionally, in order to enhance the visualization of the results, the costmaps2D are displayed in critical points such as turns or obstacles for the global and local trajectories. Additionally, it has been done a study of CPU usage of each module in all the scenarios in order to measure the impact of overloading the system.

6.4.1 Scenario I: Betancourt - Sabatini

The use case scenario is located between buildings Betancourt and Sabatini and the vehicle navigates from point A to point B, displayed in figure 6.11. The vehicle is able to make a plan that avoids the central garden and is able to navigate without collisions.

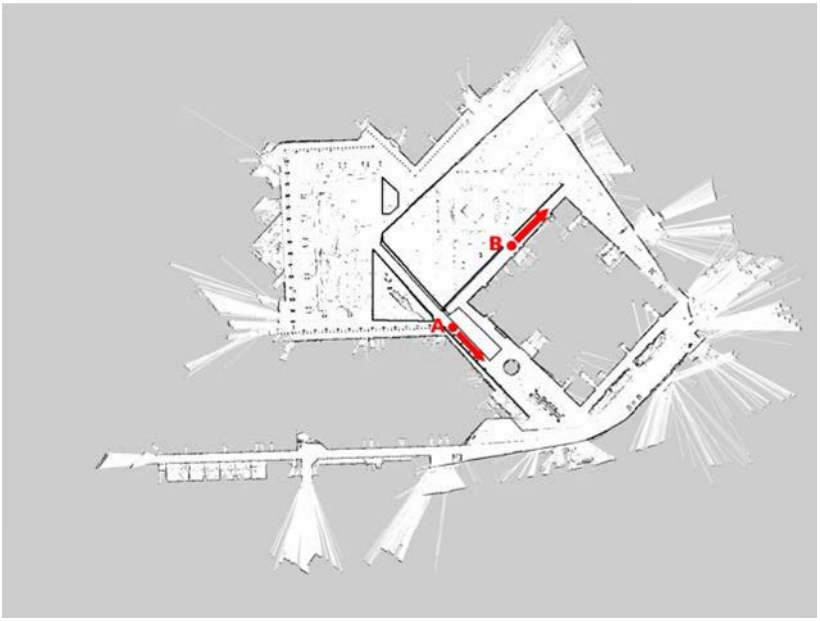


Fig. 6.11 Map with the starting point A and goal point B.

For further understanding, figure 6.12, displays the sequence of movements of the vehicle in the trajectory from left to right and from top to bottom. Blue line denotes the global path which is recalculated at 1Hz, red line denotes the local trajectory which is working at the average rate of 4Hz. The moving axis around the figure denotes de odometry point for the vehicle *rear_wheel_axis*.

In the first turn, figure 6.13 shows in detail the decision of the vehicle to make a turn to the right in order to face the curve in a better direction.

Figure 6.14 shows the CPU load for each critical navigation process and the total amount of CPU used for all of them. In figure 6.15 is possible to appreciate the influence of the extra modules and the image processing in the CPU performance due to the high computational requirement for the same modules for navigation. In both cases, the load of the processes is similar but for the overload CPU, the *icab reconfigure* (GUI) and *odometry manager* are consuming much more CPU due to the image processing display in the case of GUI and the extra modules for the odometry.

Additionally, figure 6.16 shows the CPU load for image processing and the total of image processing alone. The objective to analyze only the image processing is to show quantitatively, the computational load done in a computer without GPU (Graphical Processing Unit) which is intended to alleviate the CPU load working with threads in parallel and optimized for image processing.

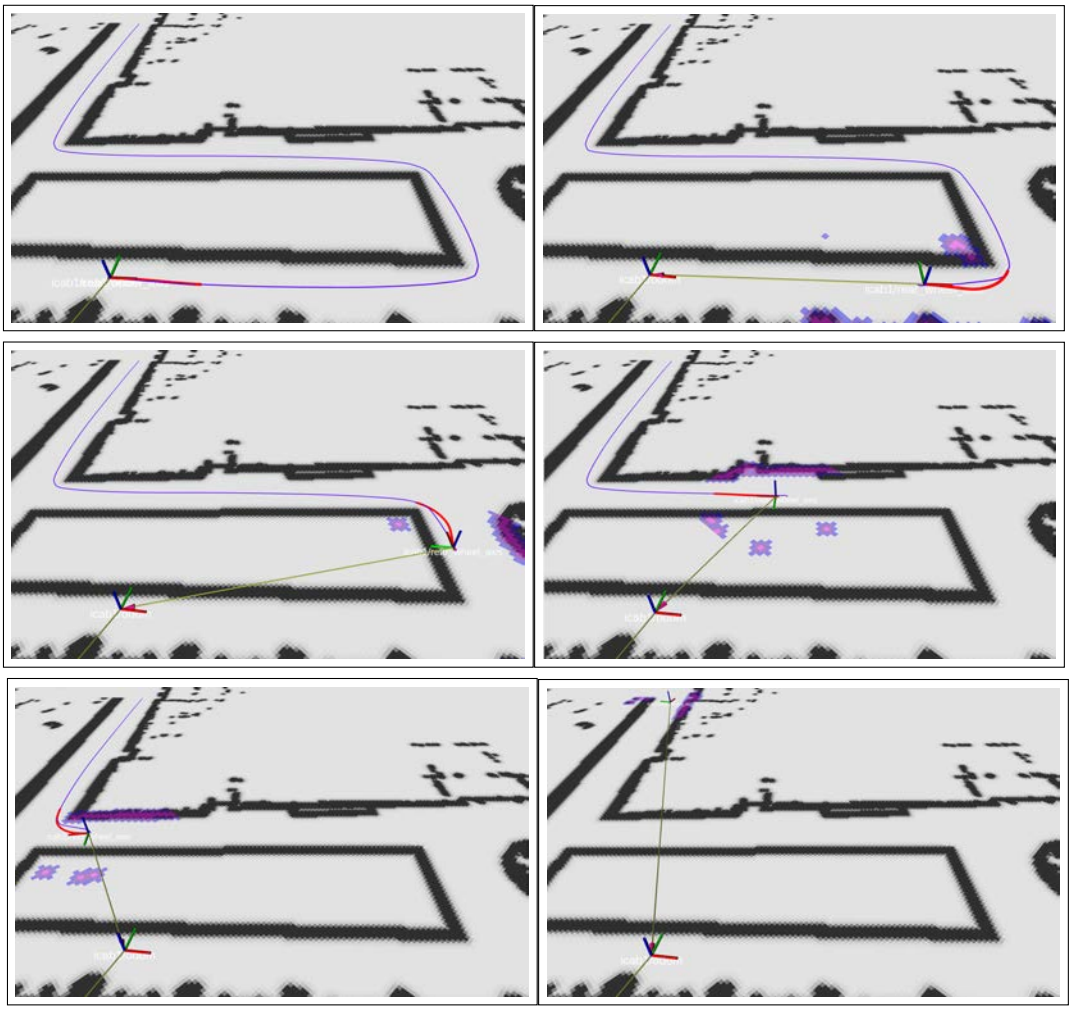


Fig. 6.12 Sequence Betancourt - Sabatini navigation.

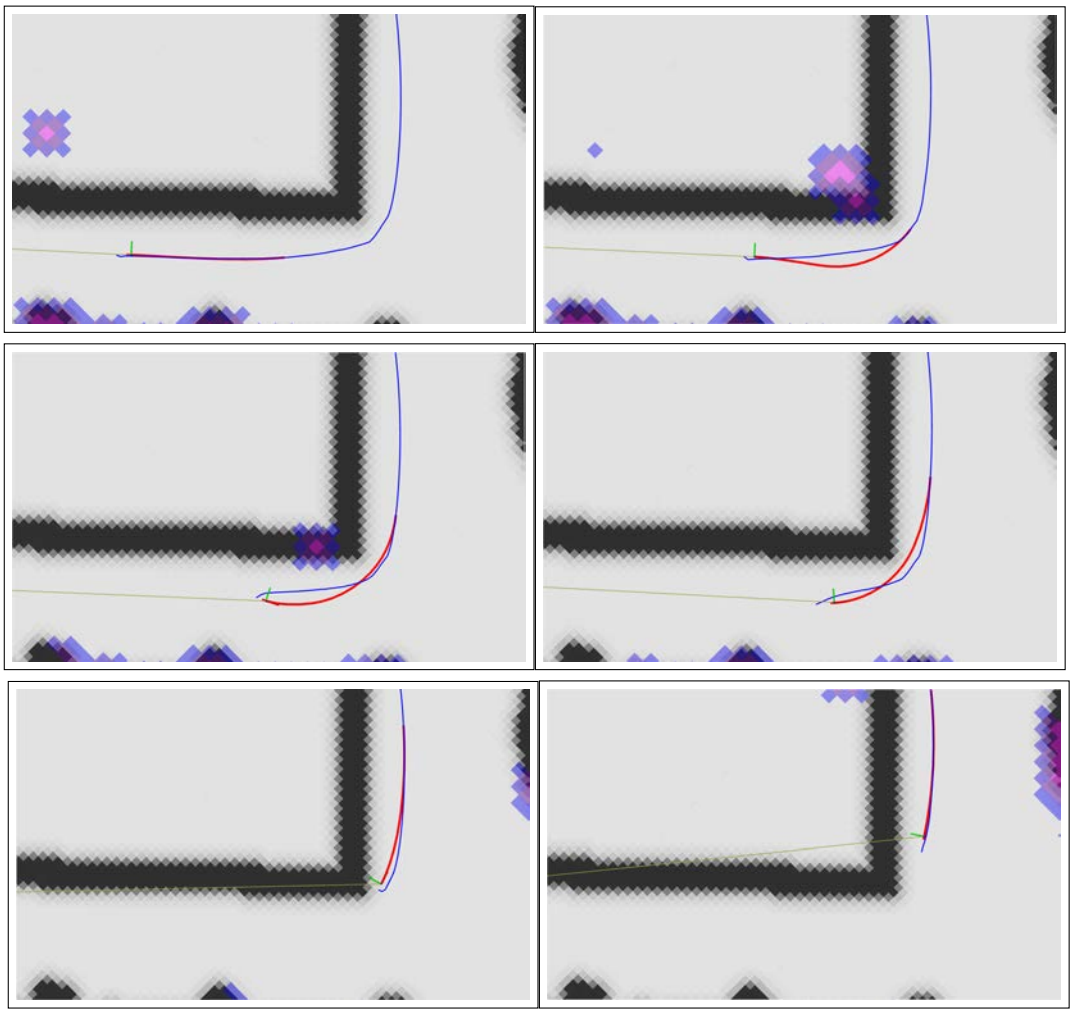


Fig. 6.13 Sequence Betancourt - Sabatini first turn in detail.

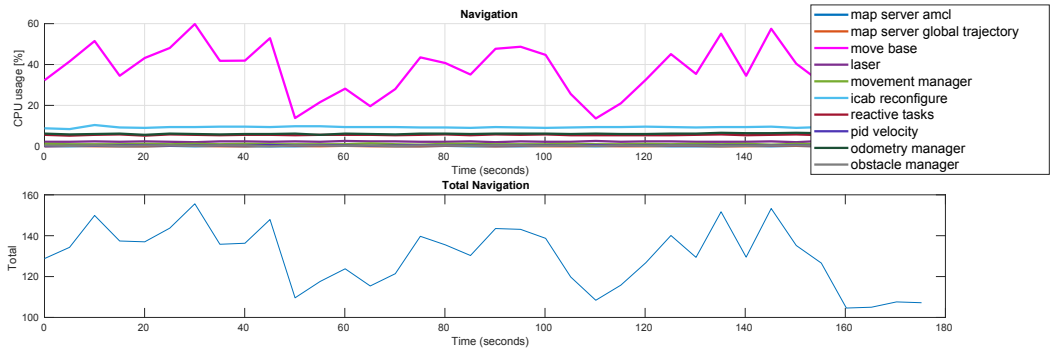


Fig. 6.14 CPU load for minimal navigation configuration.

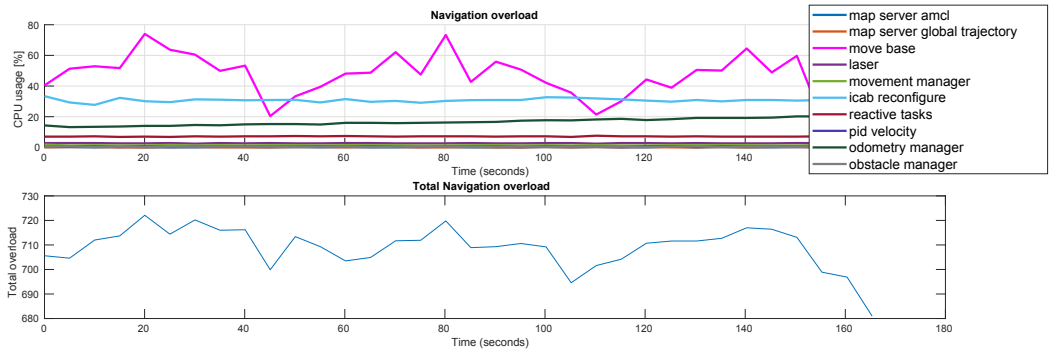


Fig. 6.15 CPU load for navigation with extra modules and image processing.

Finally, there is necessary to study the impact of the main CPU overloaded in the NUC computer. Figures 6.17 and 6.18 shows the performance of the NUC with the same nodes running in both cases, when the vehicle is configured for navigation and when the vehicle is overloaded with the image processing and extra modules.

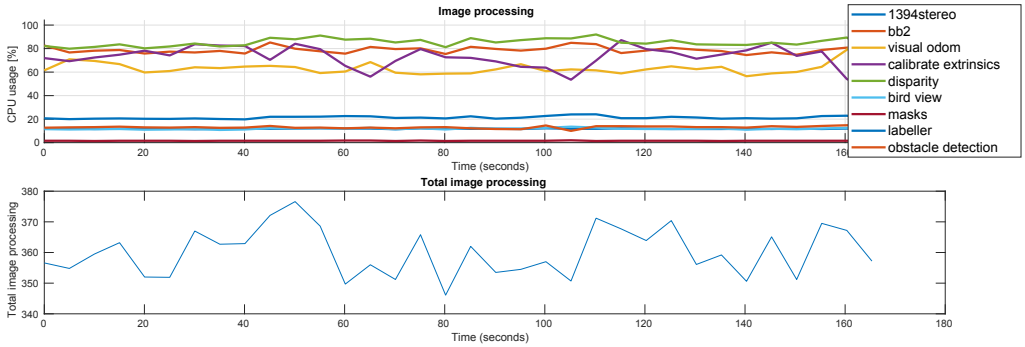


Fig. 6.16 CPU load for image processing.

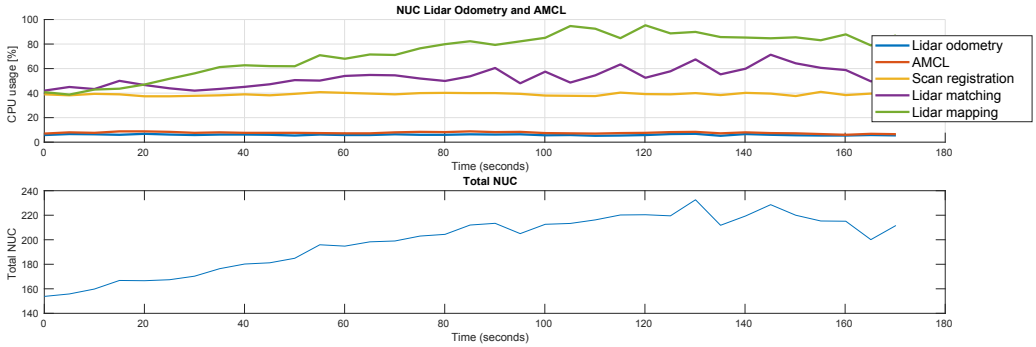


Fig. 6.17 CPU load in the Auxiliary NUC computer with minimal configuration in main computer.

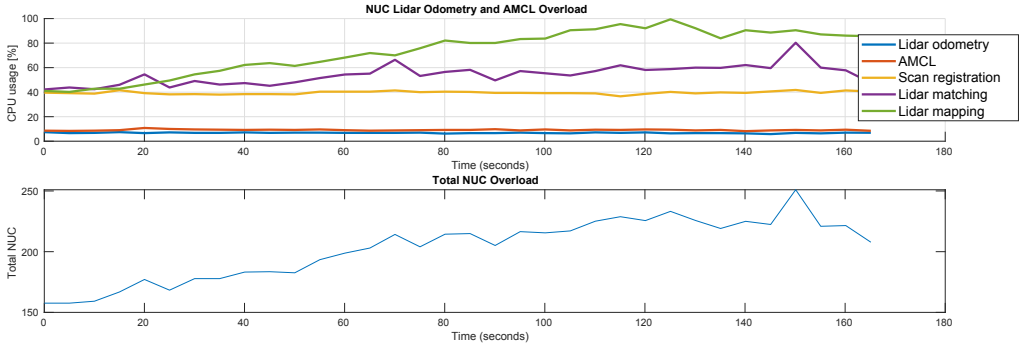


Fig. 6.18 CPU load in the Auxiliary NUC computer with overload main computer.

Another study for the rate of critical shared messages has been done in both cases, when the vehicle is overloaded with image processing and when is working with the minimum configuration for navigation. These critical messages are the rate of creation of the local costmap2D from the laser, the creation of the local trajectory, the initial pose estimation, and recalculation for global odometry and the local odometry done by the lidar. All of them are measured in Hertz in order to compare both cases and check the influence. Additionally, the boxplot for standard deviation and variance is shown on the right of each measurement.

Figure 6.19 shows the AMCL pose message published by the package AMCL which is running in the auxiliary NUC CPU. A priory it should not be affected by the overload of the main computer but this is not the case because the ROS core master is running in the main CPU and is in charge to manage the communications between the nodes, that affects slightly to the sharing rate for AMCL messages making the rate decreasing and rising the outliers.

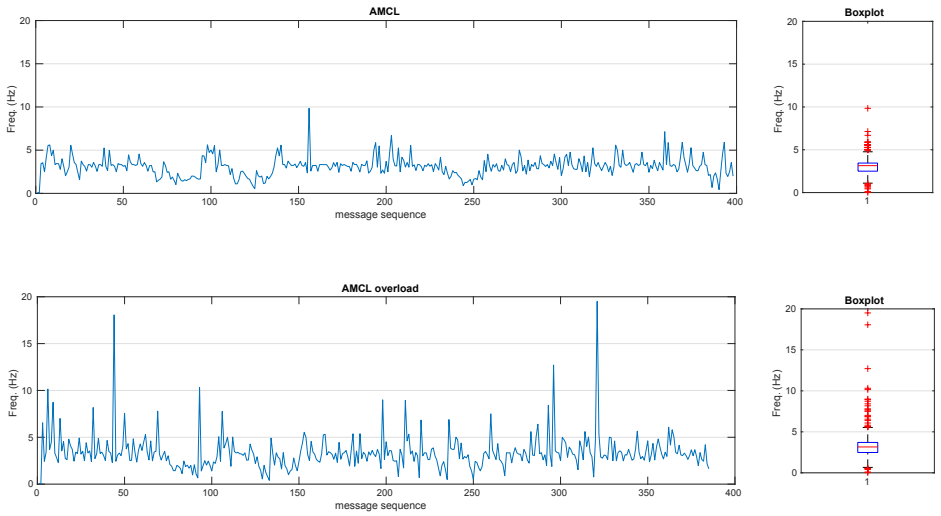


Fig. 6.19 AMCL frequency message generation analysis.

Figure 6.20 shows the lidar odometry message shared at a specific rate of 10Hz with variations in the overloaded computer. These variations make the odometry slightly worst due to the decreasing rate of publishing (measures under 10 Hz are highly repeated in overloaded lidar odometry).

Figure 6.21 shows the local costmap2D generation. In this case, the costmap2D published are divided into the whole costmap2D and the costmap2D updates. This is a feature in the costmap2D package in order to save bandwidth. The difference resides that the costmap2D publish the whole map and the costmap2D updates publish only the variations from the last costmap2D. When these variations are many enough, the whole costmap2D is published

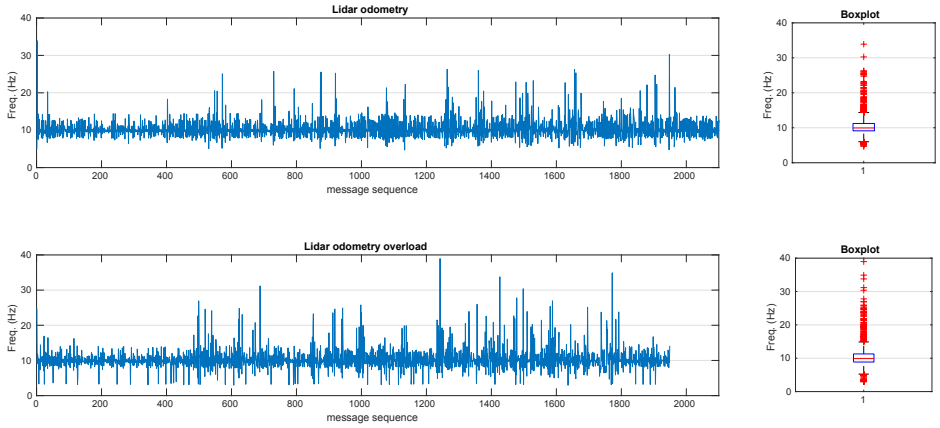


Fig. 6.20 Lidar odometry frequency message generation analysis.

again. In order to measure both, the published of costmap2D and the updates, it is taken both published times and measure the rate when any of them are published. The comparison between the minimum configuration and the overloaded computer is that the local costmap2D decreases the rate in the second case, which was expected.

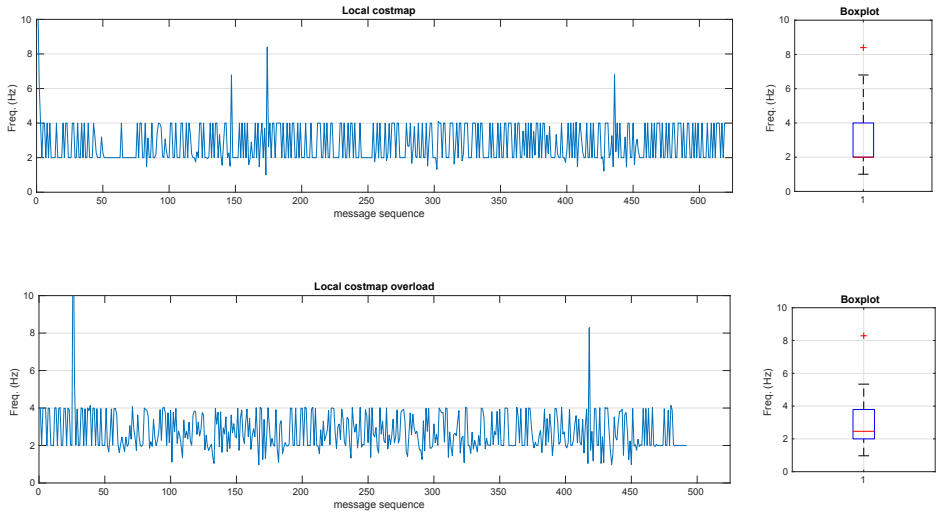


Fig. 6.21 Local costmap2D frequency message generation analysis.

For the local trajectory, which is the most critical part in the navigation stage, the rate oscillates between 2 and 4 Hz (250ms) which is enough to navigate at slow speed. For fast speed, it is necessary to increase the control loop which highly depends on the costmap2D creation. Regarding the overloaded CPU for trajectory generation, it is possible to appreciate

the rise of standard deviation and variance in the rate. That means that the rate decreases which in some events, the rate goes under 2Hz which it is dangerous due to the fact that the vehicle could not react in time to an imminent hazard.

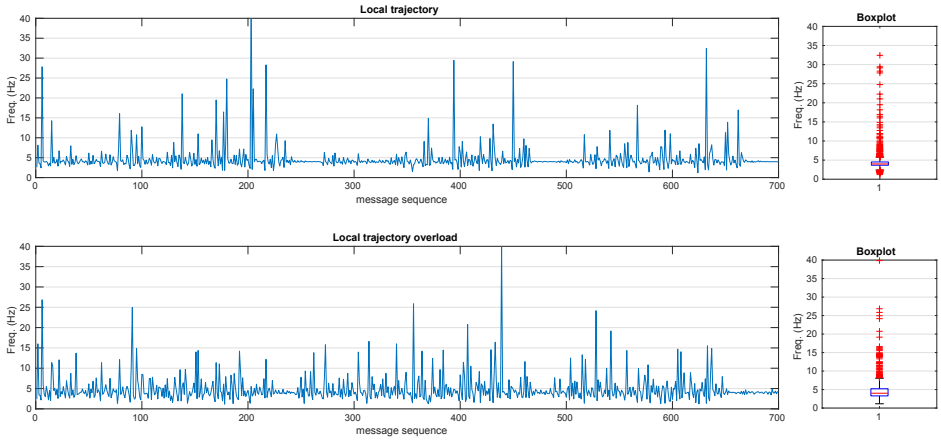


Fig. 6.22 Local trajectory frequency message generation analysis.

Regarding the creation of the commands from the TEB local planner and the low-level control, it is still a doubt if an overloaded computer interferes with the control. In order to do this study, has been taken the reference velocity and steering angle as the output of the local planner and the current velocity and steering angle.

Figures 6.23 and 6.24 shows in both cases in the same experiment the current velocity and the reference velocity. It is not possible to appreciate differences between both, hence, it is possible to assume that there is no influence to the low-level control in an overloaded computer. This conclusion has not be mistaken with correct or wrong decisions of the local plan which could be wrong generated but the low-level control could be working as expected.

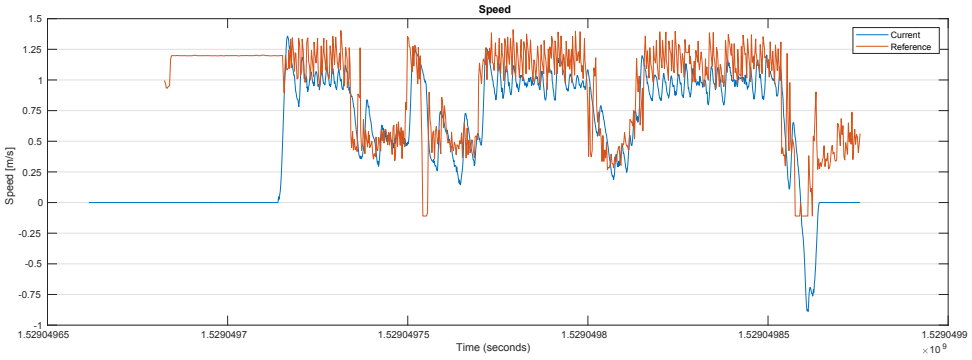


Fig. 6.23 Low level speed control performance with minimal configuration I.

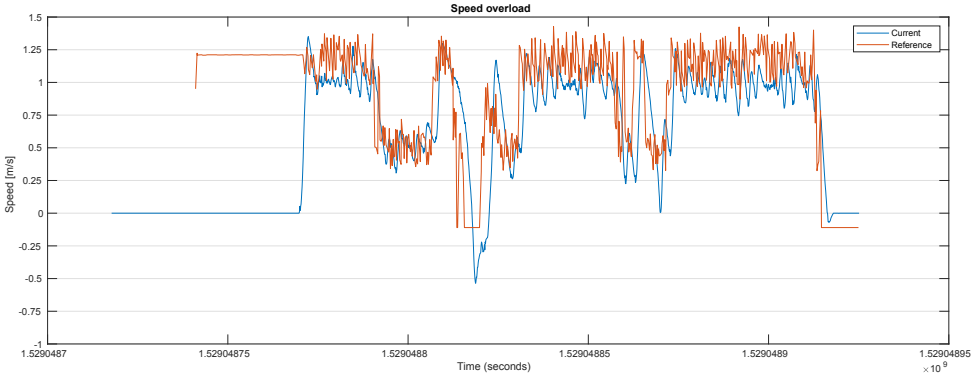


Fig. 6.24 Low level speed control performance with image processing configuration I.

The same study has been done for steering angle with similar results shown in figures 6.25 and 6.26

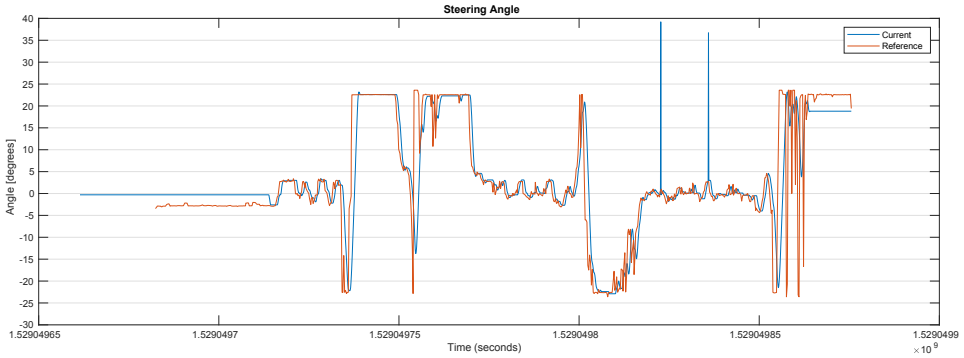


Fig. 6.25 Low level steering angle control with minimal configuration I.

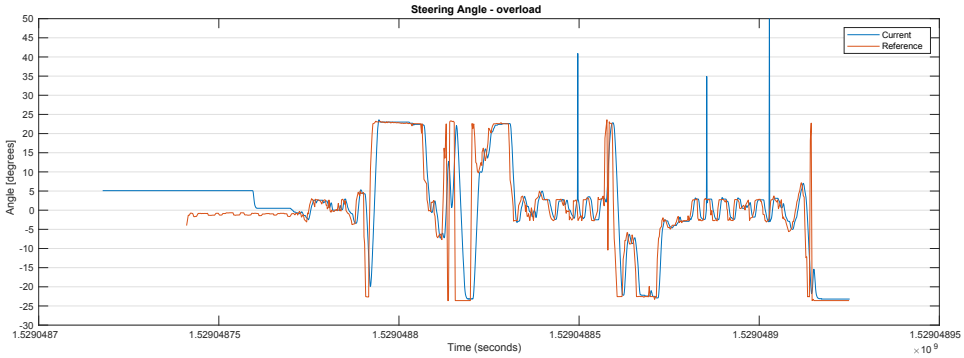


Fig. 6.26 Low level steering angle control performance with image processing configuration I.

6.4.2 Scenario II: Sabatini - Betancourt

This scenario takes place in the same spot as in the first scenario but from the Sabatini building to the Betancourt building. Figure 6.27 displays the origin point A and the goal point B, both with orientations. The experiment has been realized in order to take more information about the rates of the messages for different scenarios in order to gather more data and check if the system is reliable for different places.

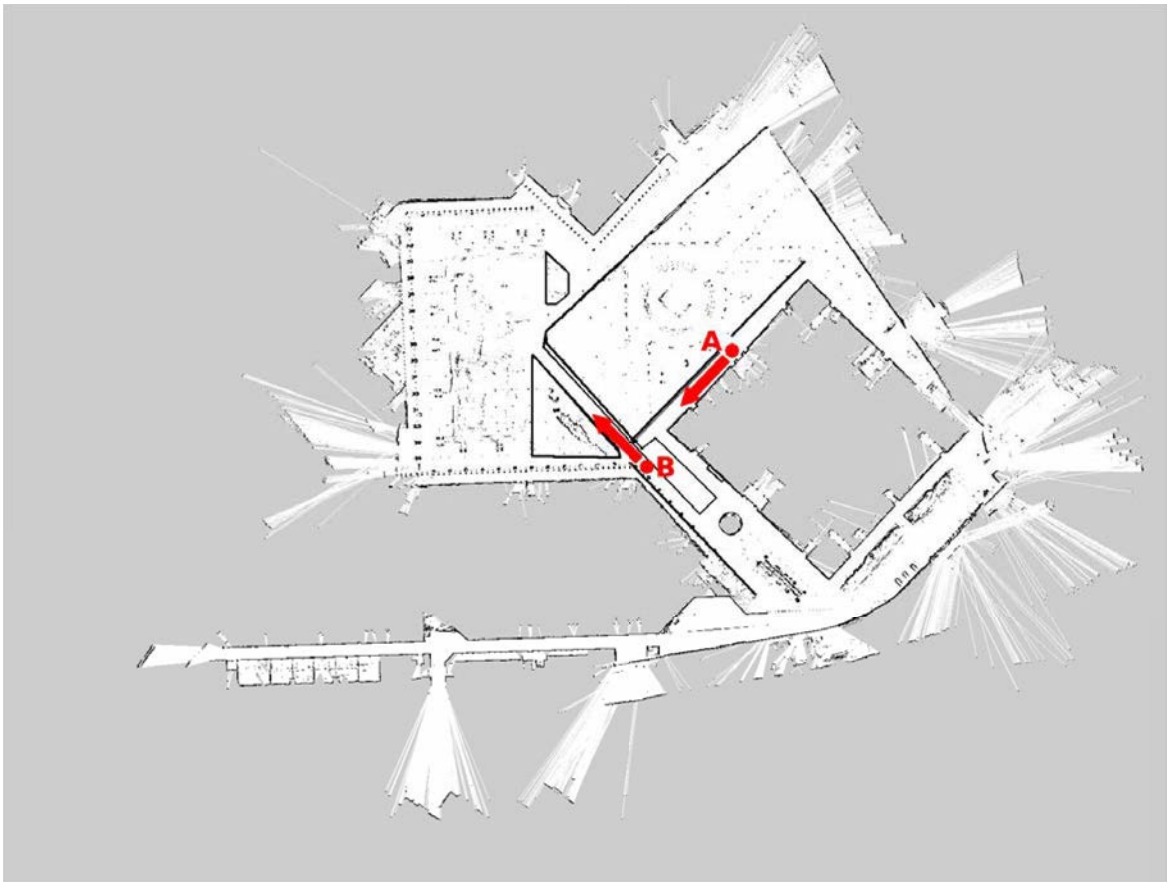


Fig. 6.27 Map with the starting point A and goal point B.

The sequence of movements, in this case, goes from Sabatini building to Betancourt building showed in figure 6.28.

The CPU load study is displayed in figures 6.29, 6.30, where again, the influence of an overloaded computer is consuming between 650 and 700% of the total of 800% where some processes are slowed down in the message generation rate. Figure 6.31, shows the computational power requirement only for image processing and finally, the NUC computer is showed in 6.32 for the minimum configuration and 6.33 for the navigation with image processing and extra modules.

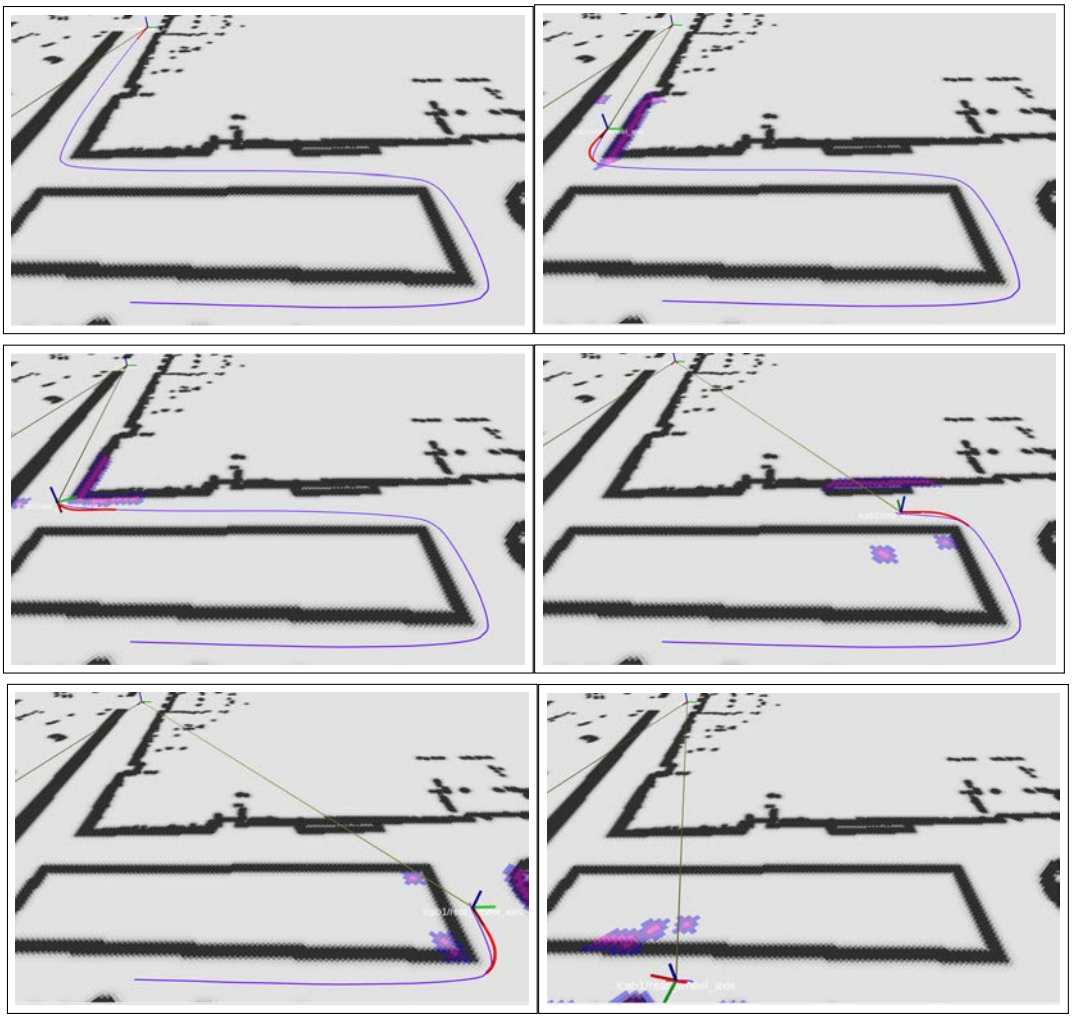


Fig. 6.28 Sequence Sabatini - Betancourt navigation.

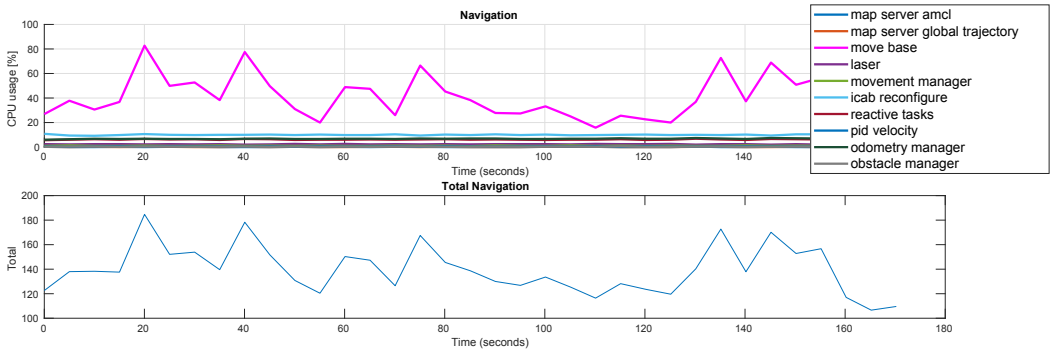


Fig. 6.29 CPU load for minimal navigation configuration.

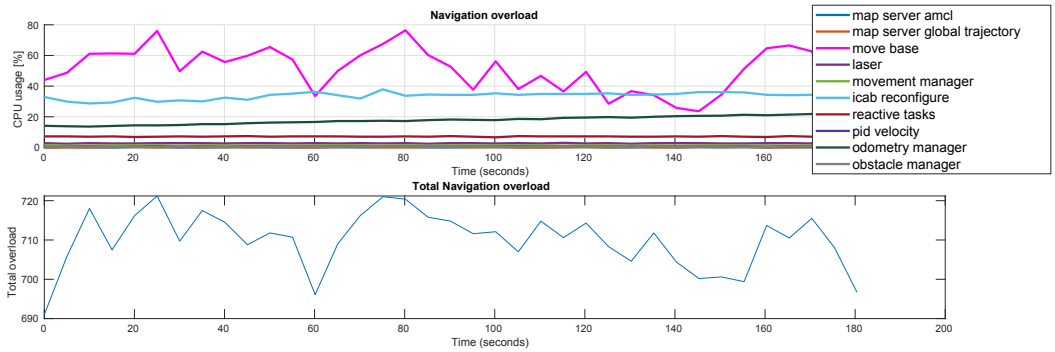


Fig. 6.30 CPU load for navigation with extra modules and image processing.

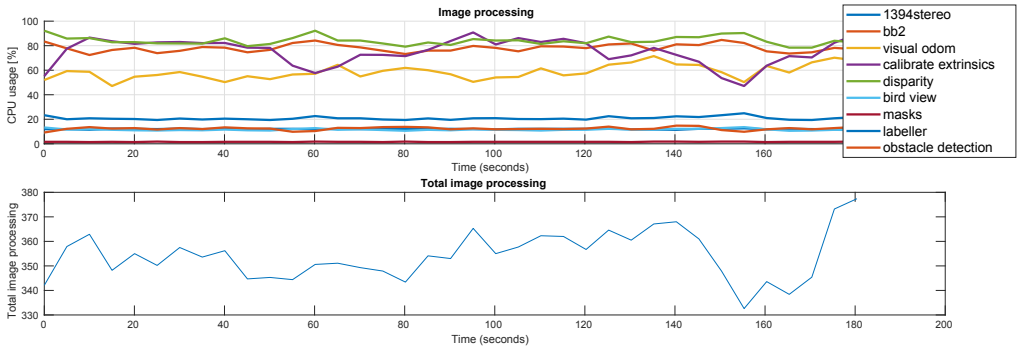


Fig. 6.31 CPU load for image processing.

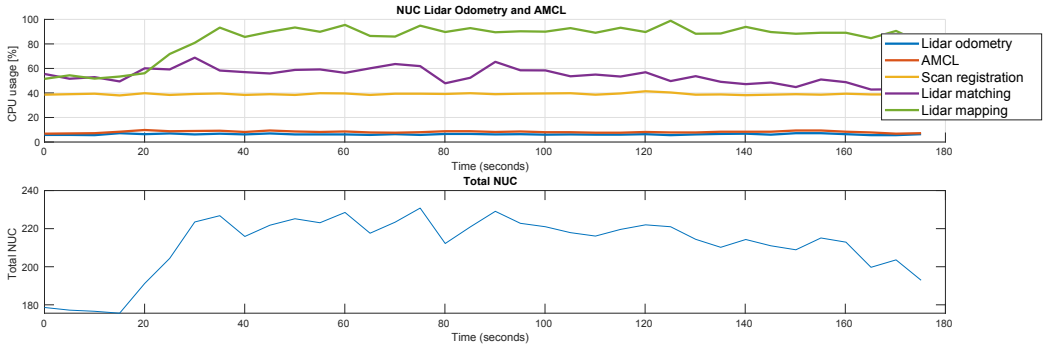


Fig. 6.32 CPU load in the Auxiliary NUC computer with minimal configuration in main computer.

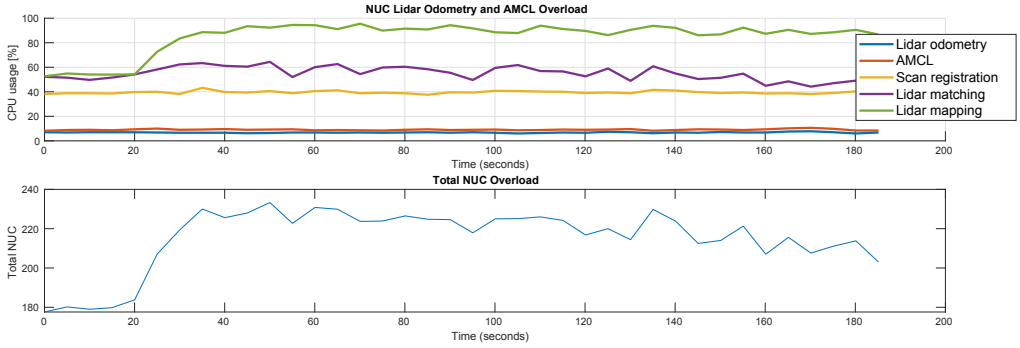


Fig. 6.33 CPU load in the Auxiliary NUC computer with overload main computer.

The frequency of the messages published by the critical navigation modules is again tested in this scenario and figures 6.34, 6.35, 6.36 and 6.37. The result are similar, where the influence of an overloaded computer decrease the publish rate of critical messages such as local trajectories or costmap2D.

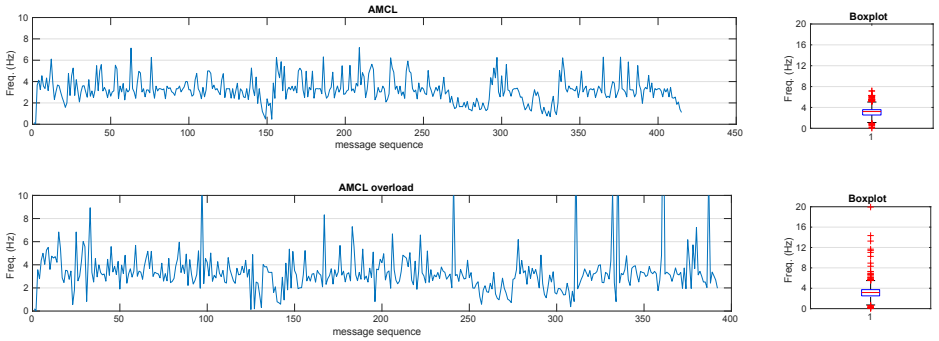


Fig. 6.34 AMCL frequency message generation analysis.

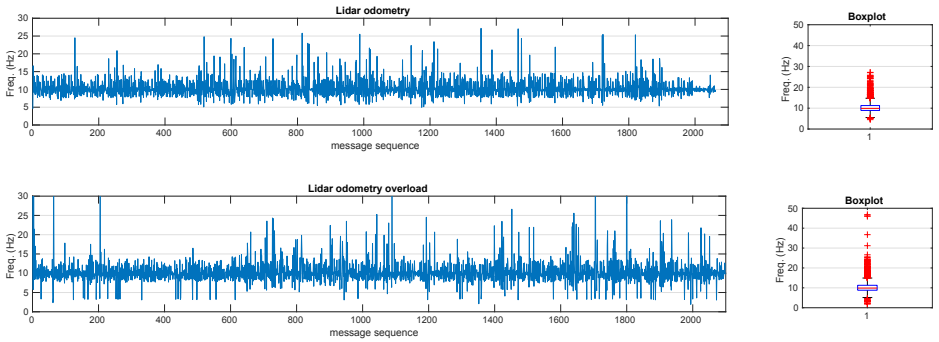


Fig. 6.35 Lidar odometry frequency message generation analysis.

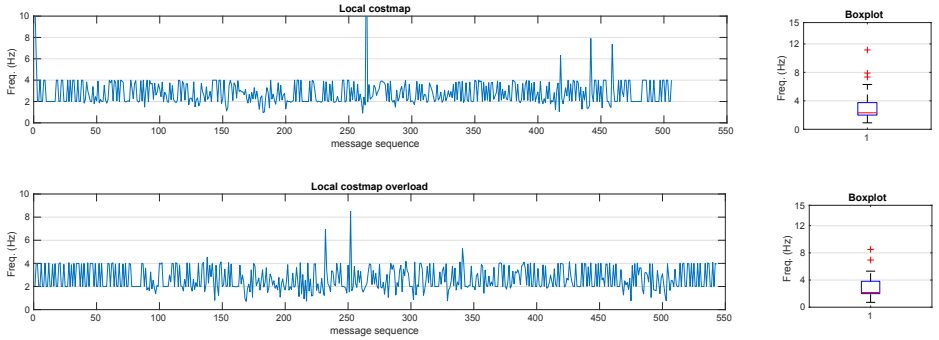


Fig. 6.36 Local costmap2D frequency message generation analysis.

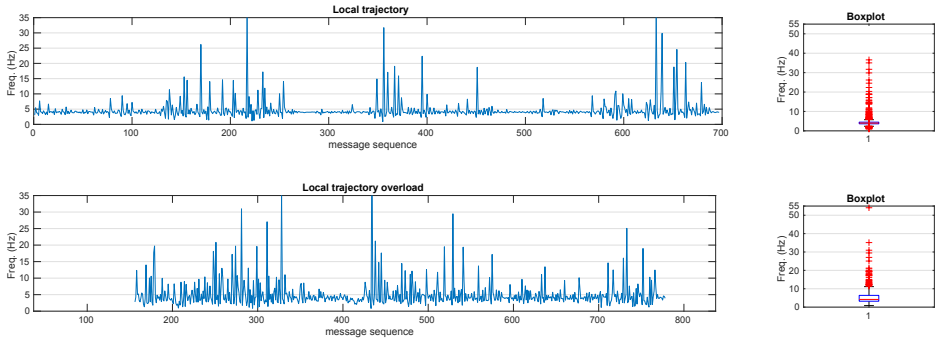


Fig. 6.37 Local trajectory frequency message generation analysis.

Finally, for the controller, figures 6.38, 6.39, 6.40 and 6.41, shows the behavior in the trajectory as the command generation from the TEB local planner and the current value for speed and steering angle. In both cases, the low-level control is still able to follow the commands without any influence of an overloaded computer.

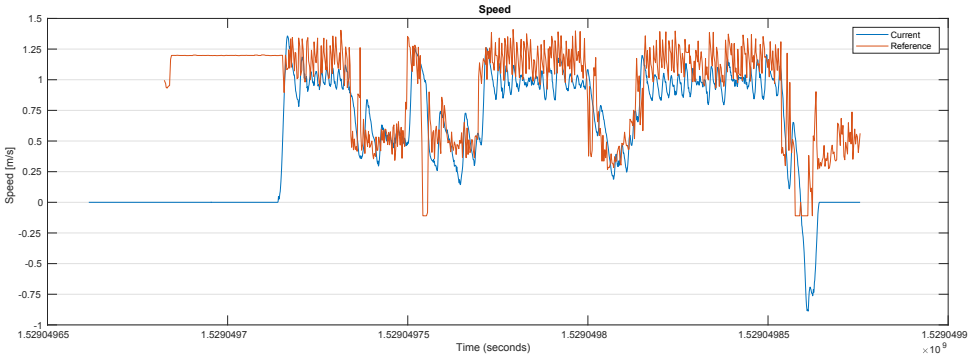


Fig. 6.38 Low level speed control performance with minimal configuration II.

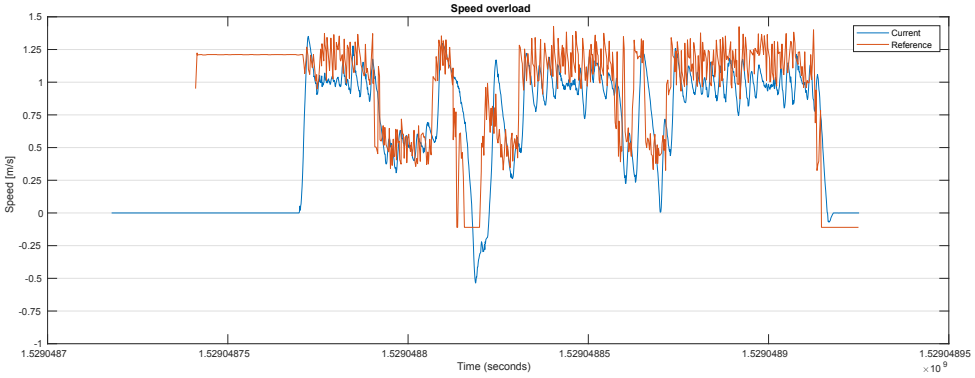


Fig. 6.39 Low level speed control performance with image processing configuration II.

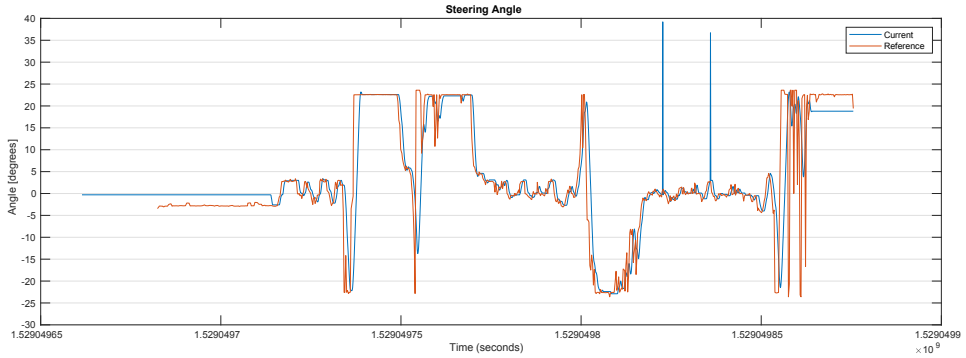


Fig. 6.40 Low level steering angle control performance with minimal configuration II.

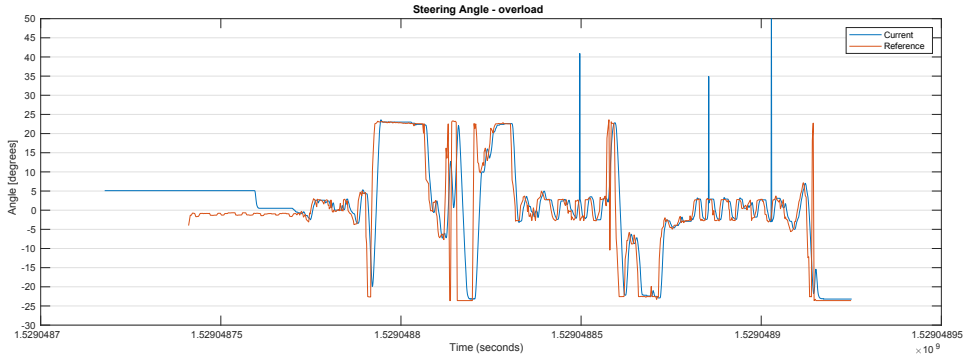


Fig. 6.41 Low level steering angle control performance with image processing configuration II.

6.4.3 Scenario III: Betancourt - Library

In this use case, the vehicle is located at the entrance of Betancourt building (point A) and receives the order to navigate to the library (point B), figure 6.42. The vehicle starts the navigation by pressing a button in the touchscreen from the GUI and is able to avoid collisions and reach the goal without trouble. This scenario is interesting for being a crowded environment where the trajectories generated by the vehicle are in direct collision with the pedestrian trajectories. Furthermore, the endpoint at the library is a hard goal to achieve for the vehicle with 8 meters of looking forward in the local trajectory because the global plan does not take into account the orientation but the local plan does, hence, the final maneuver space is only 8 meters wide which makes it impossible to reach in some situations and vehicle needs to go backwards. Due to the fact that the vehicle requires at least one second in order to execute the emergency brake for imminent collision, in this scenario and in the next one, it was required the operator intervention to avoid collisions. Despite the human intervention in this critical situations, the system was able to detect and send the brake order but the mechanical brake is not fast enough.

The sequence of movements, in this case, goes from Sabatini building to Betancourt building showed in figure 6.43.

Following the same scheme, figures 6.44, 6.45 shows the influence of running the extra modules (figure 6.46). The CPU load in the auxiliary NUC computer are shown in 6.47 for the minimum configuration and 6.48 for the navigation with image processing and extra modules. Even when the vehicle is at full CPU load, is able to navigate but with a poorer performance which means more oscillations in the trajectory generation.

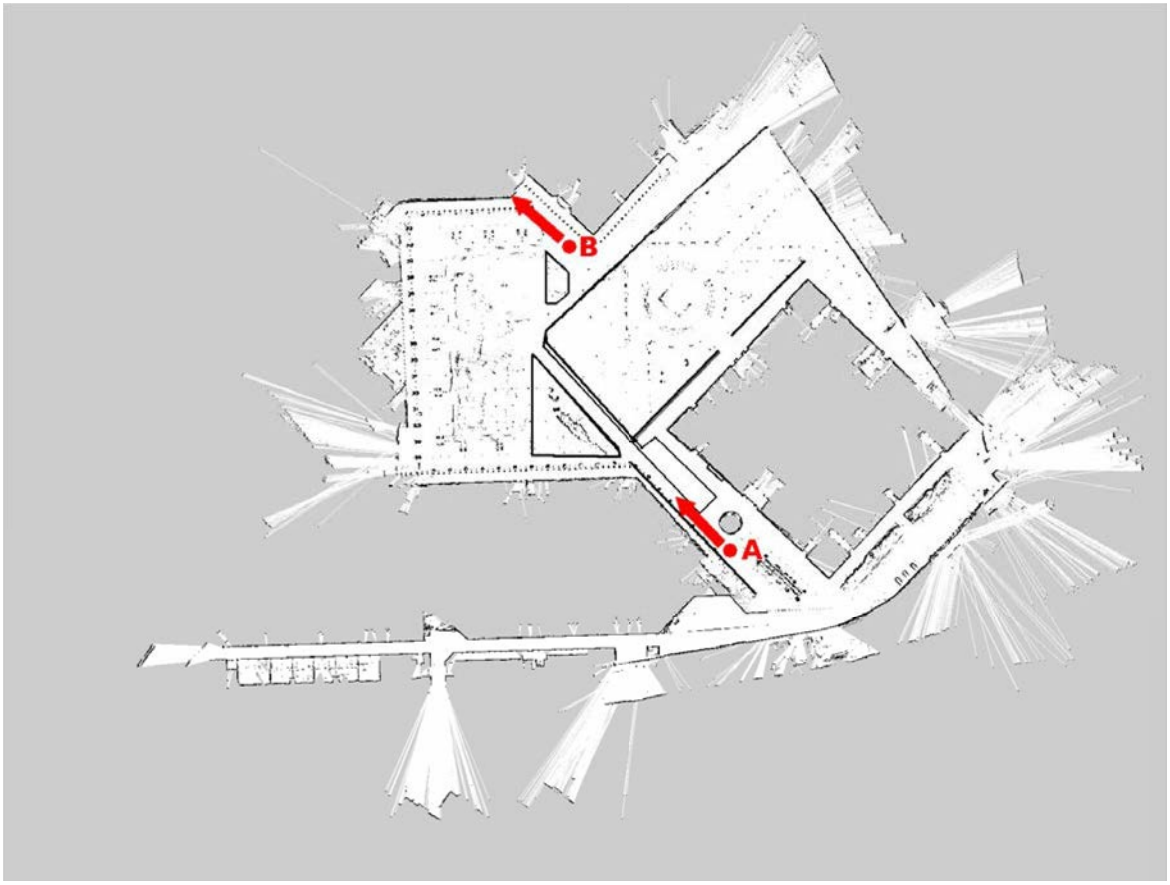


Fig. 6.42 Map with the starting point A and goal point B.

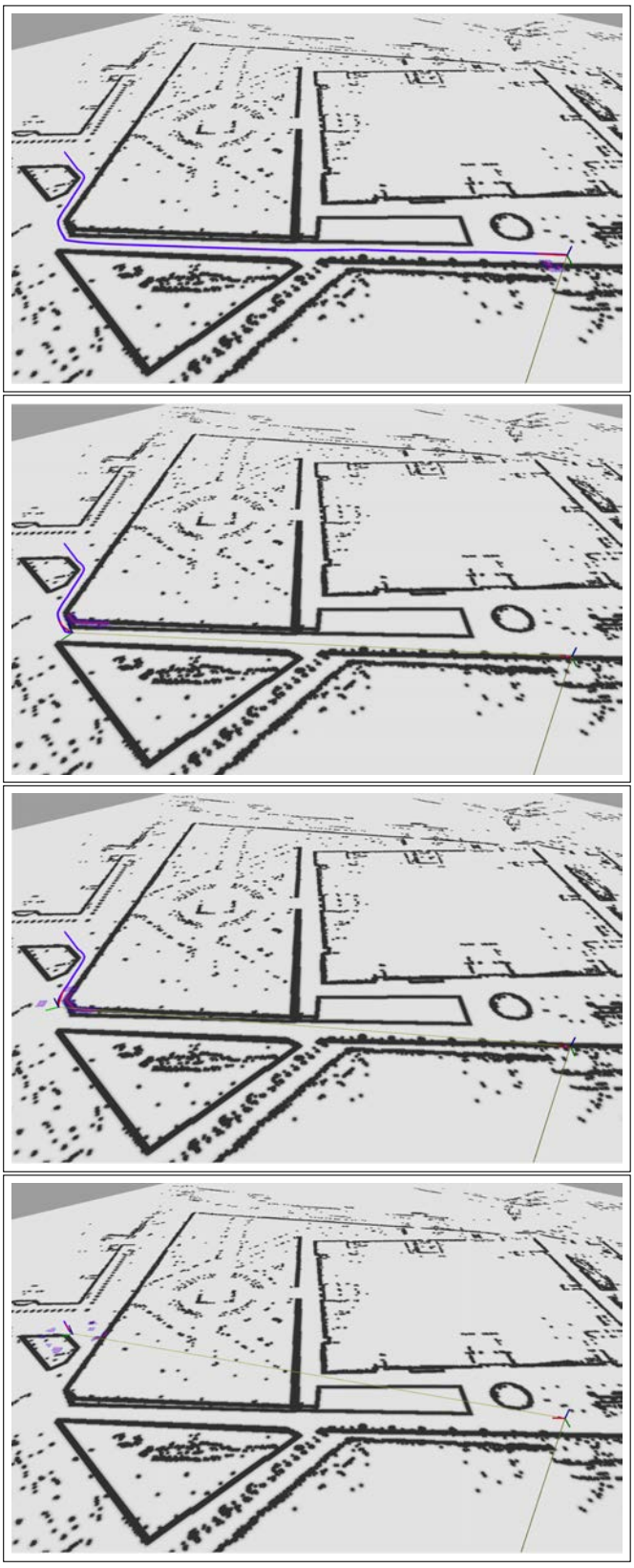


Fig. 6.43 Sequence Library - Betancourt navigation.

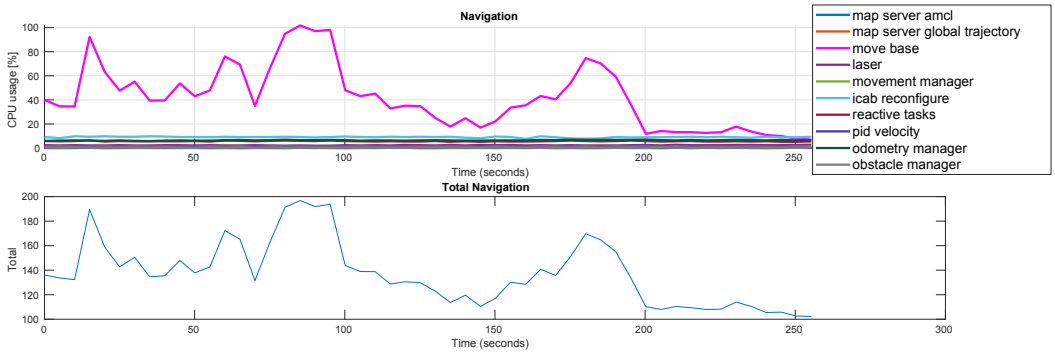


Fig. 6.44 CPU load for minimal navigation configuration.

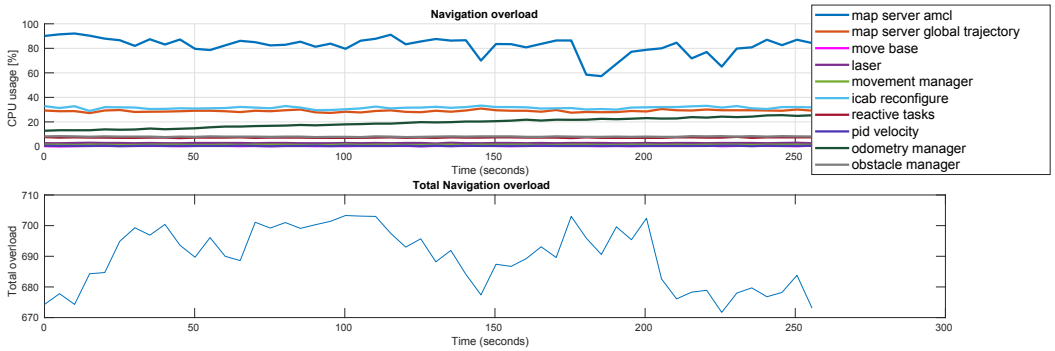


Fig. 6.45 CPU load for navigation with extra modules and image processing.

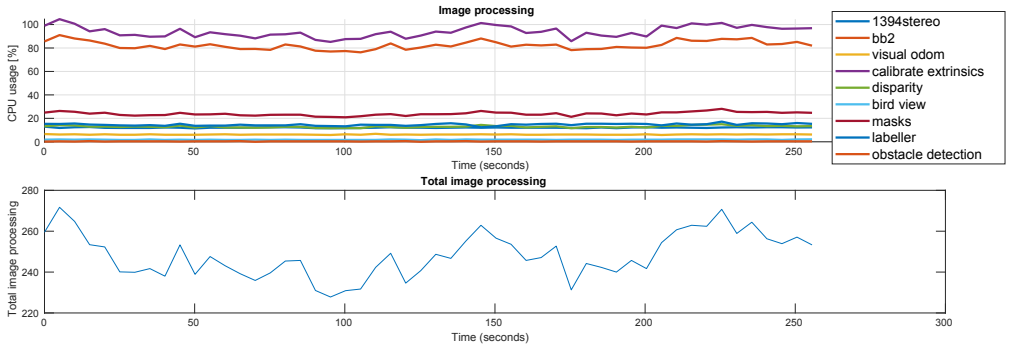


Fig. 6.46 CPU load for image processing.

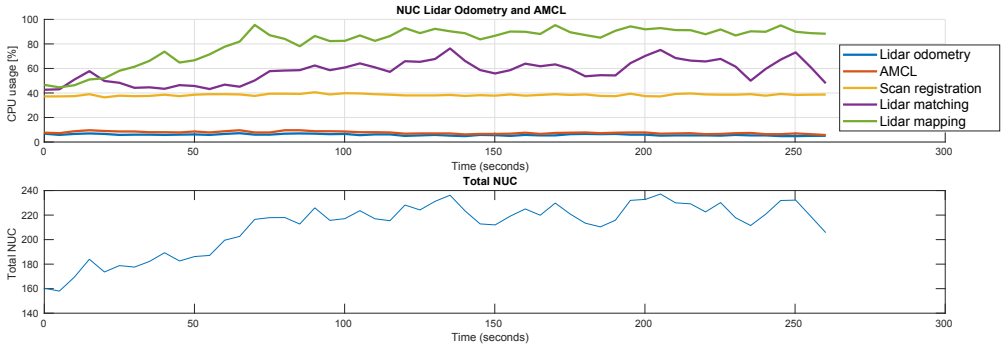


Fig. 6.47 CPU load in the Auxiliary NUC computer with minimal configuration in main computer.

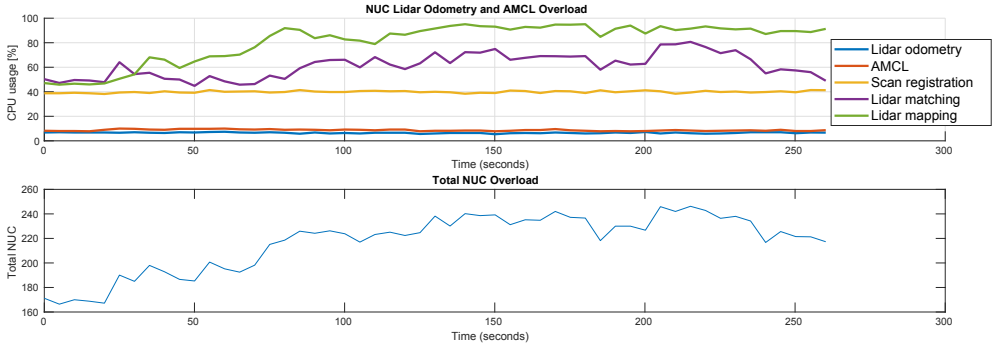


Fig. 6.48 CPU load in the Auxiliary NUC computer with overload main computer.

As seen in the previous scenarios, the critical shared messages for navigation, displayed in figures 6.49, 6.50, 6.51 and 6.52, show the decreasing of rate in local trajectory generation and local costmap2D generation.

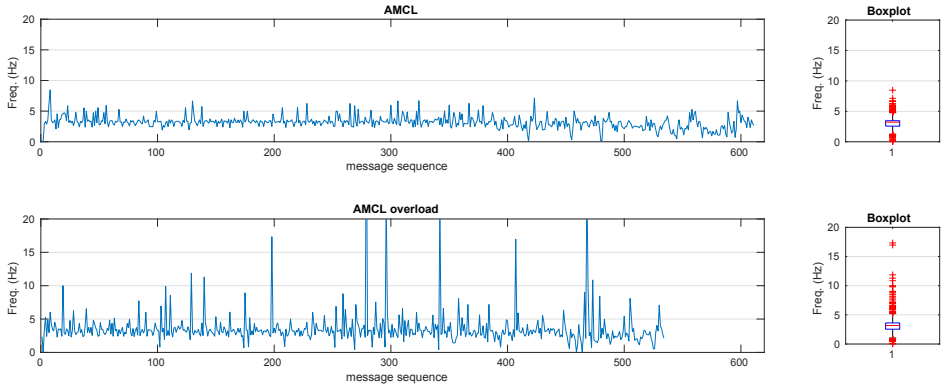


Fig. 6.49 AMCL frequency message generation analysis.

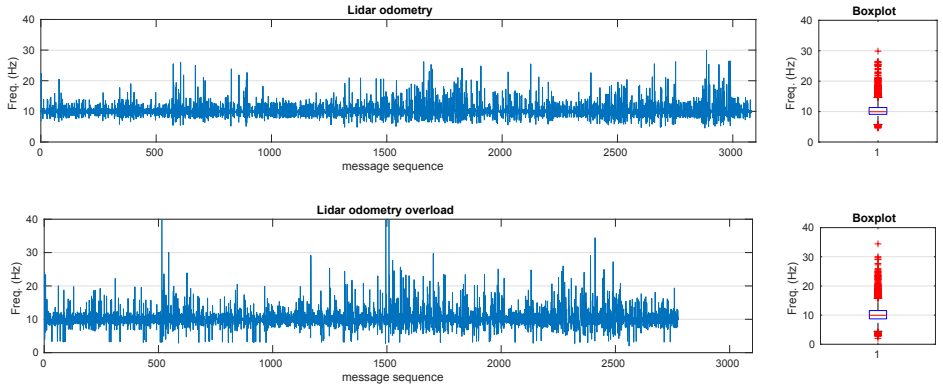


Fig. 6.50 Lidar odometry frequency message generation analysis.

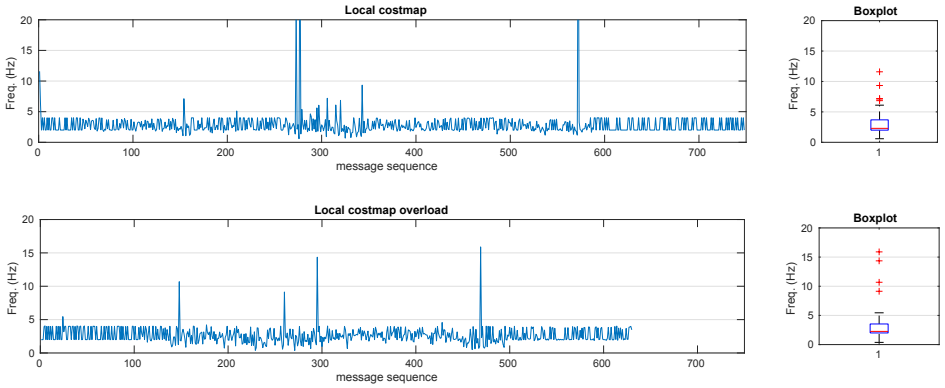


Fig. 6.51 Local costmap2D frequency message generation analysis.

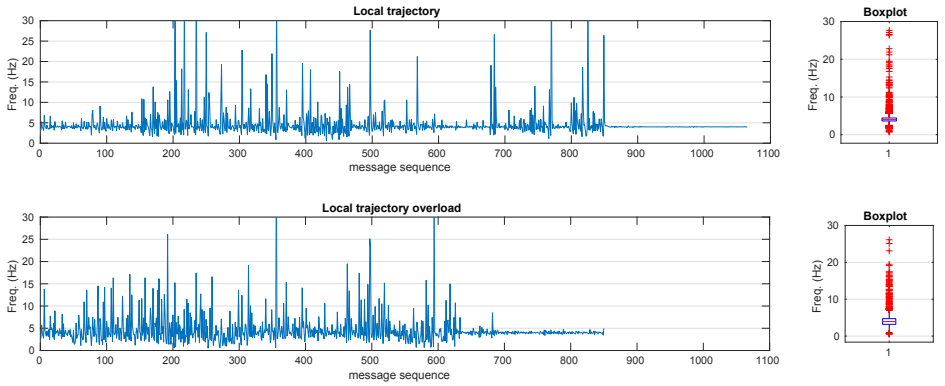


Fig. 6.52 Local trajectory frequency message generation analysis.

For the speed, figures 6.53 and 6.54, and steering angle, figures 6.55 and 6.56, the controller is able to reach the commands in both cases. Notice that the TEB local planner is publishing commands at the beginning of the experiment but the vehicle does not move until the button in the GUI is pressed.

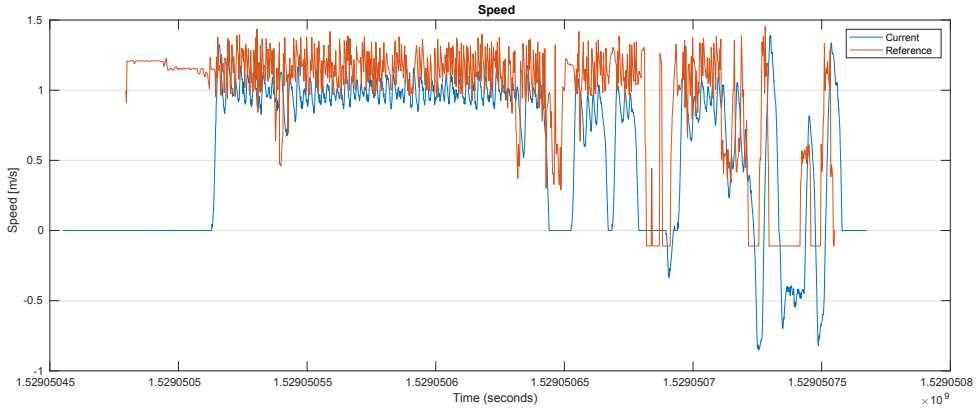


Fig. 6.53 Low level speed control performance with minimal configuration III.

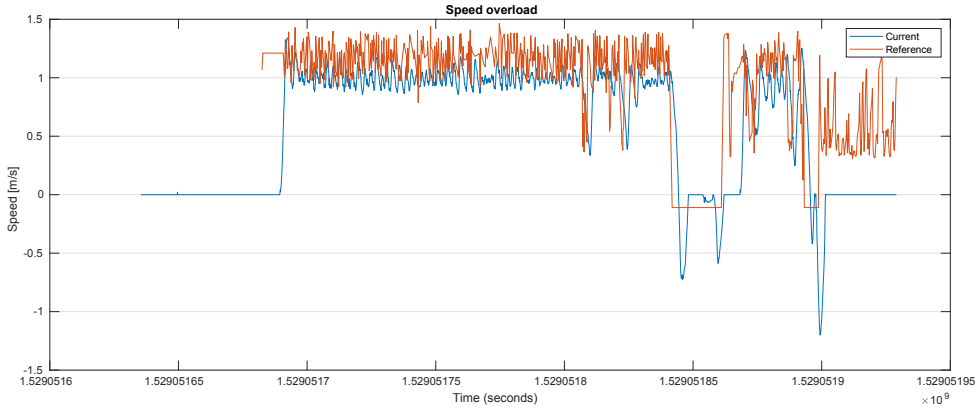


Fig. 6.54 Low level speed control performance with image processing configuration III.

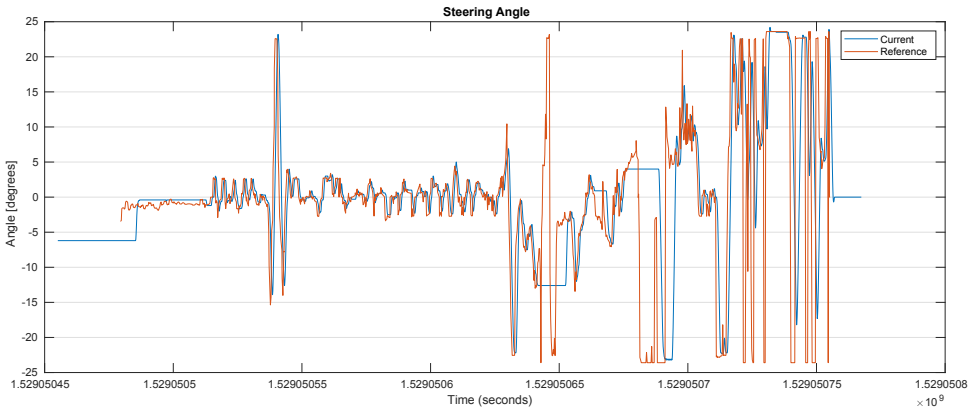


Fig. 6.55 Low level steering angle control performance with minimal configuration III.

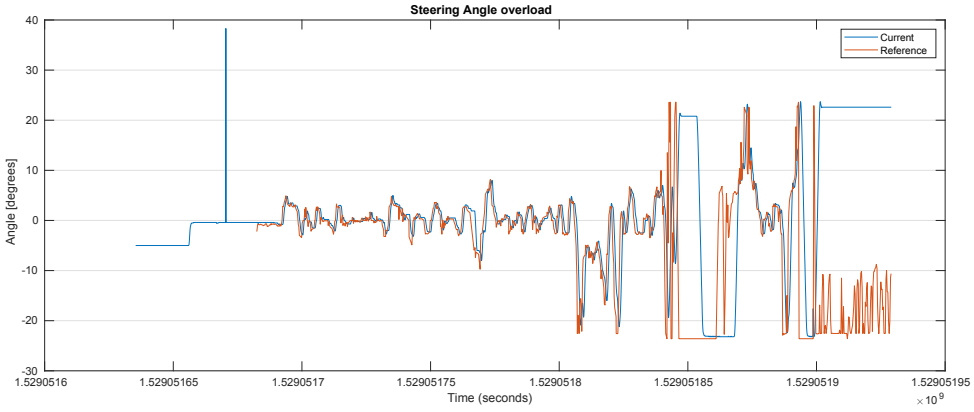


Fig. 6.56 Low level steering angle control performance with image processing configuration III.

6.4.4 Scenario IV: Library - Betancourt

In this use case, the vehicle is located at the entrance of the Library building (point A) and the goal point is in front of the Betancourt building. The importance of this scenario is the initial location for the AMCL which require matching points from the frontal laser but due to the fact that the starting point is in an open space, the matching is not accurate. Even with fewer points, the movement of the vehicle after some readings makes the AMCL more accurate because the vehicle faces the fence with more points to match.

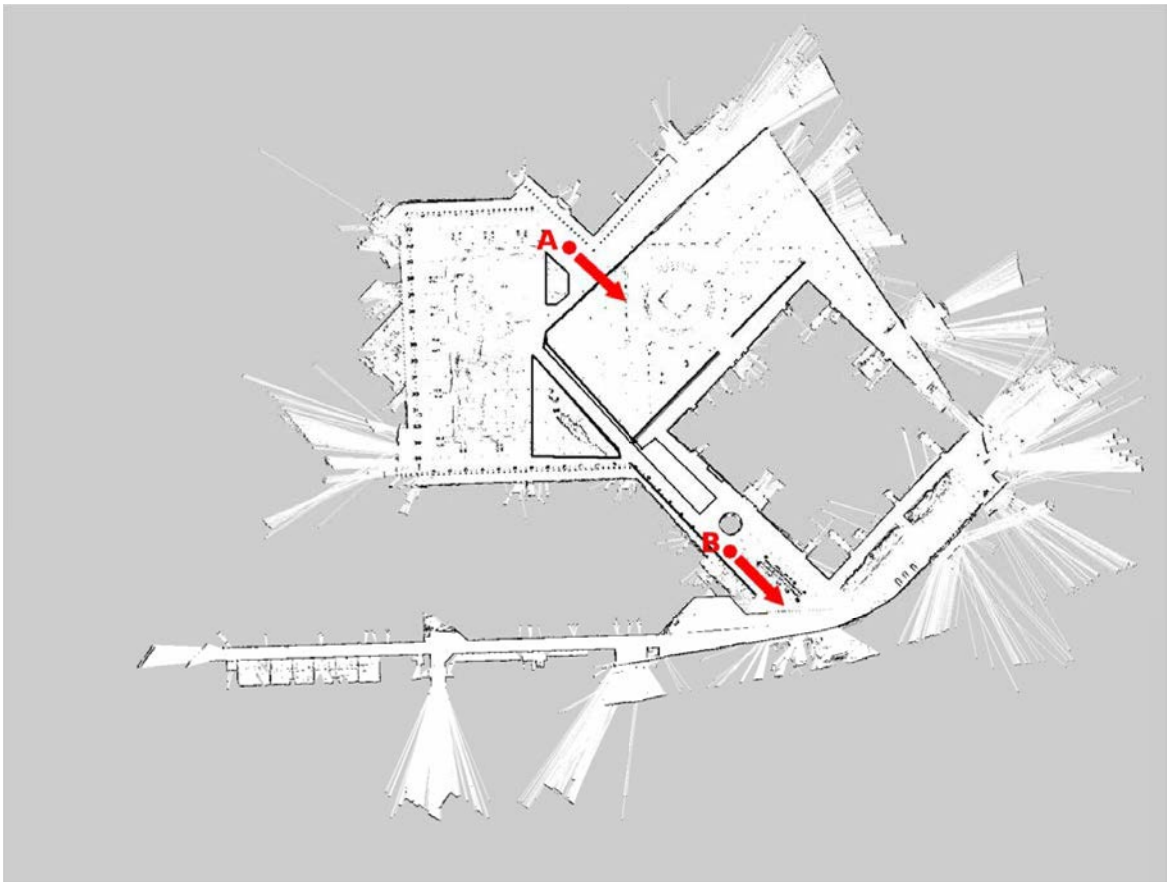


Fig. 6.57 Map with the starting point A and goal point B.

The sequence of movements, in this case, goes from Sabatini building to Betancourt building showed in figure 6.58.

Similar results have been experienced in this scenario regarding the previous one. The CPU load for the main computer in both cases are shown in figures 6.59, 6.60. The main CPU load only for image processing is shown in 6.61 which is still very high and the CPU for the auxiliary NUC in both experiments with and without image processing and extra modules are shown in 6.62 for the minimum configuration and 6.63 for the overload main CPU.

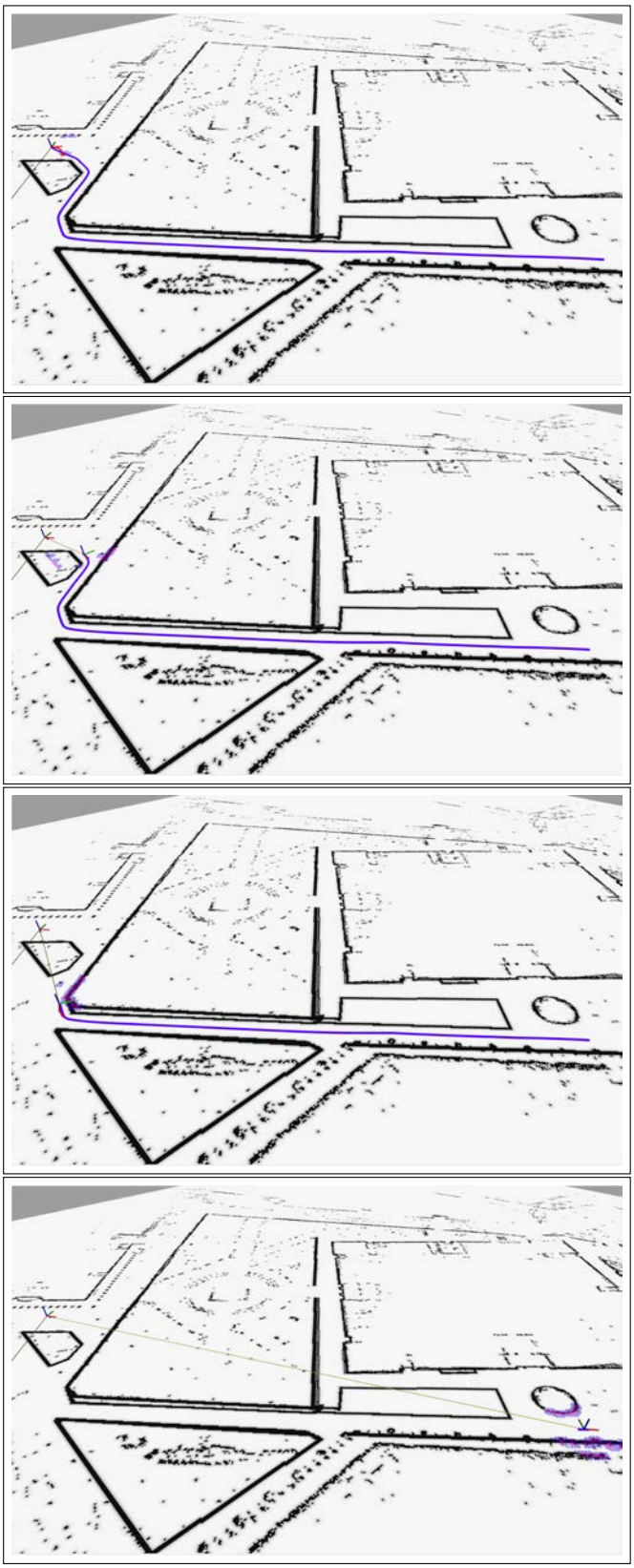


Fig. 6.58 Sequence Library - Betancourt navigation.

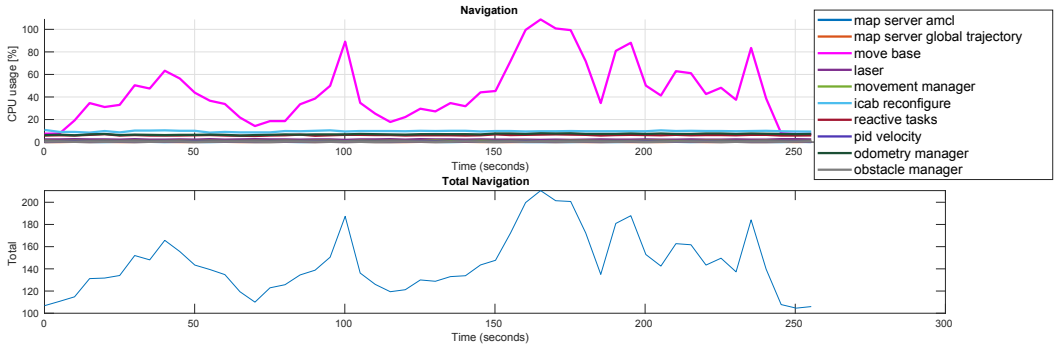


Fig. 6.59 CPU load for minimal navigation configuration.

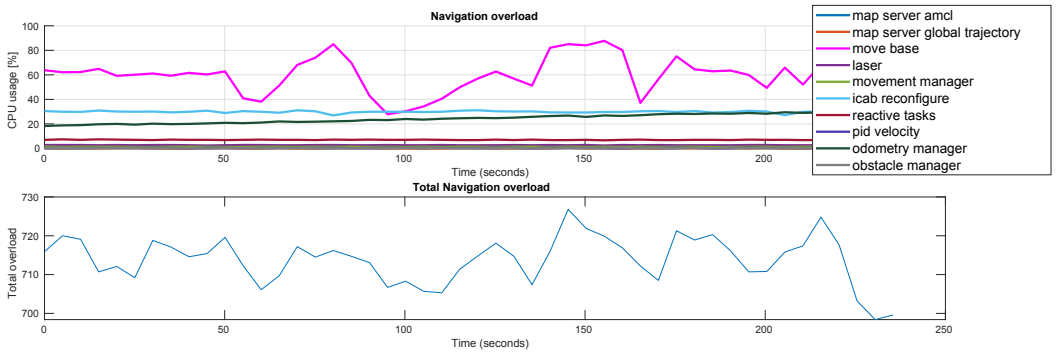


Fig. 6.60 CPU load for navigation with extra modules and image processing.

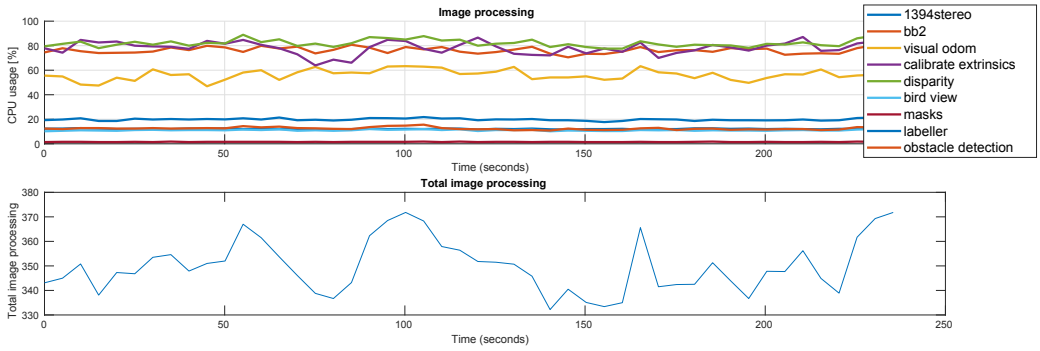


Fig. 6.61 CPU load for image processing.

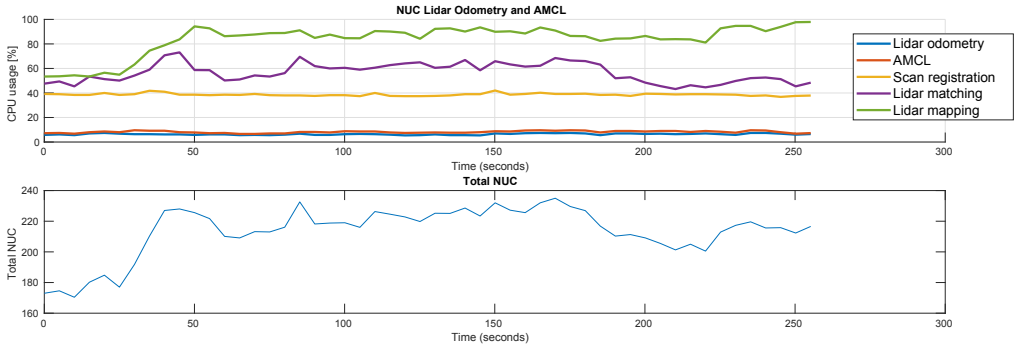


Fig. 6.62 CPU load in the Auxiliary NUC computer with minimal configuration in main computer.

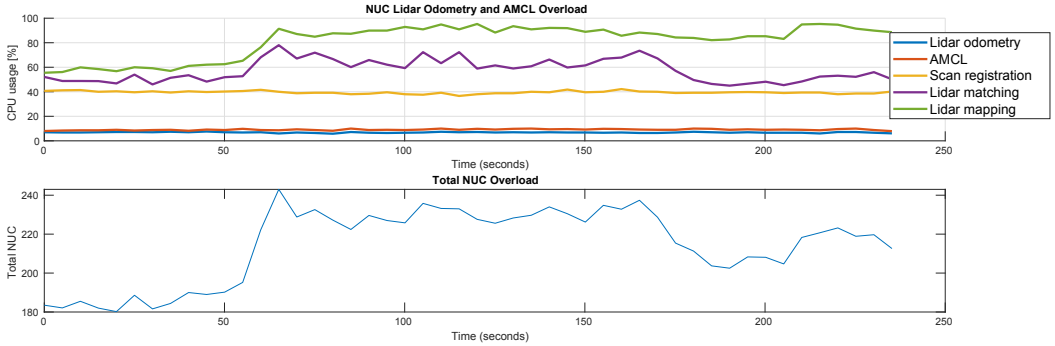


Fig. 6.63 CPU load in the Auxiliary NUC computer with overload main computer.

Again, the frequency rate of the published navigation modules drop slightly in the experiment where extra modules and image processing are running in parallel providing less accurate local trajectories (figures 6.64, 6.65, 6.66and 6.67).

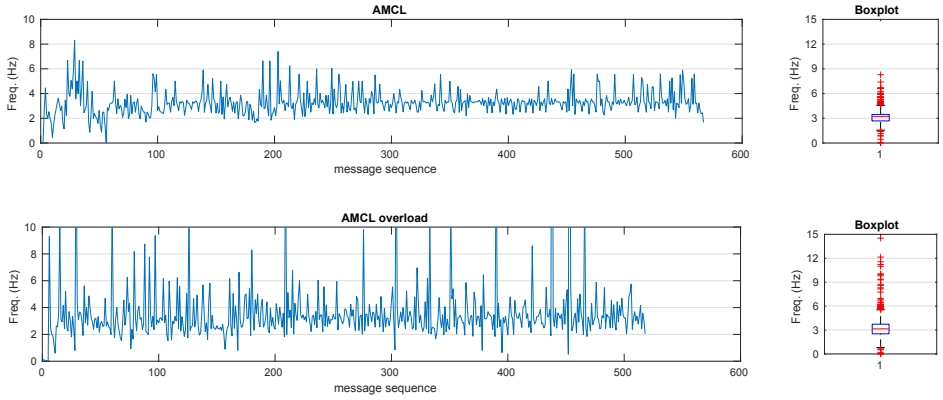


Fig. 6.64 AMCL frequency message generation analysis.

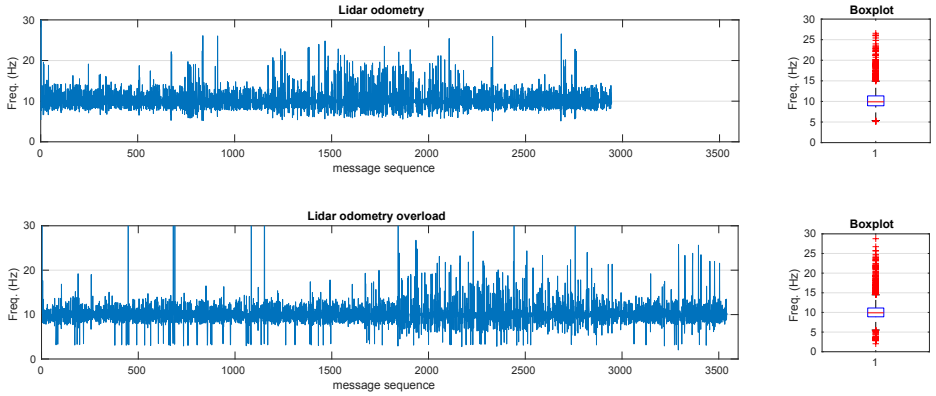


Fig. 6.65 Lidar odometry frequency message generation analysis.

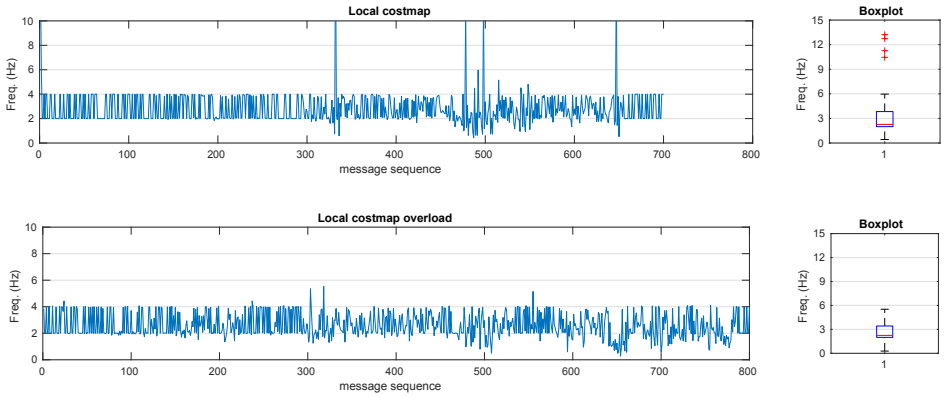


Fig. 6.66 Local costmap2D frequency message generation analysis.

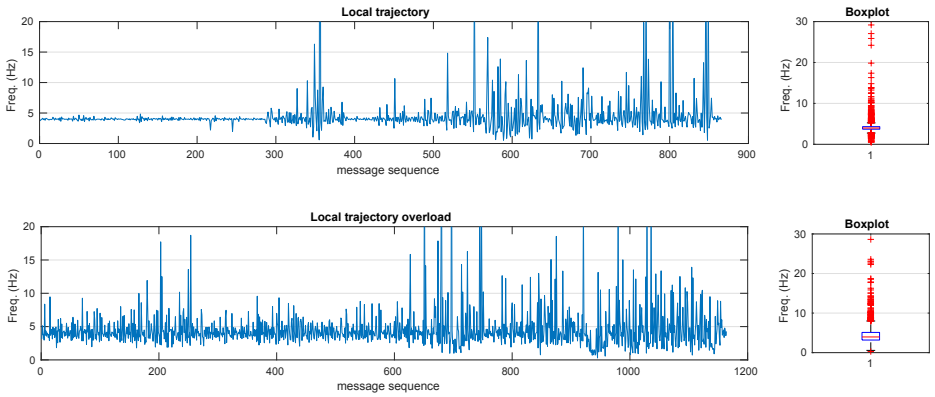


Fig. 6.67 Local trajectory frequency message generation analysis.

As in the previous experiments, the controller is still able to manage the speed and steering angle provided by the local planner, figures 6.68, 6.69, 6.70 and 6.71

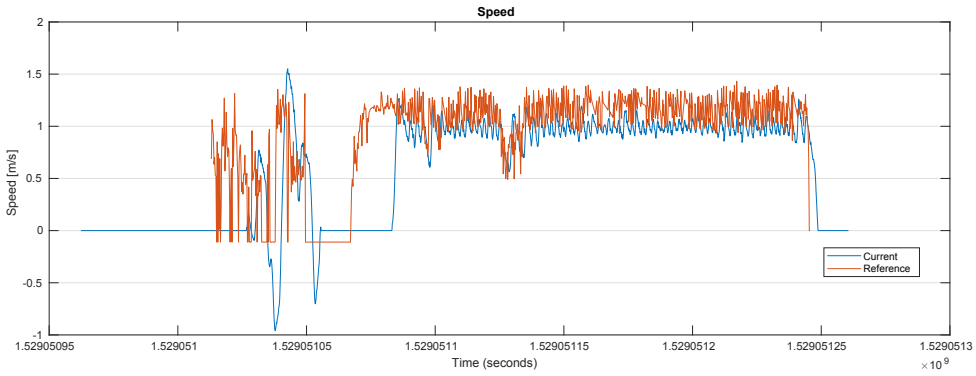


Fig. 6.68 Low level speed control performance with minimal configuration IV.

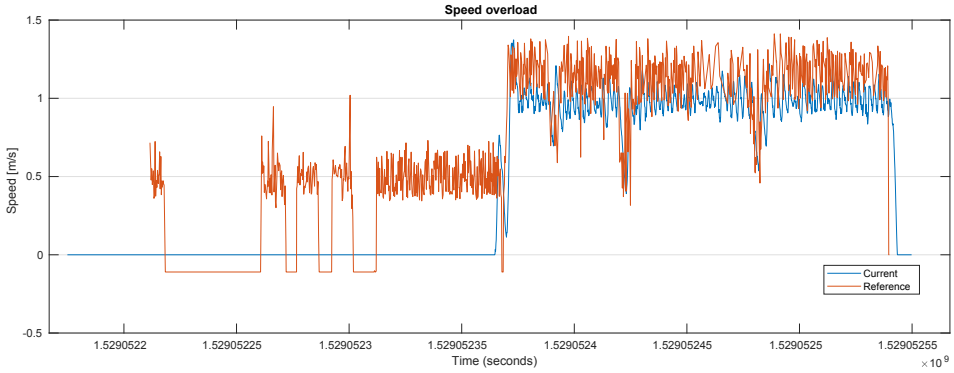


Fig. 6.69 Low level speed control performance with image processing configuration IV.

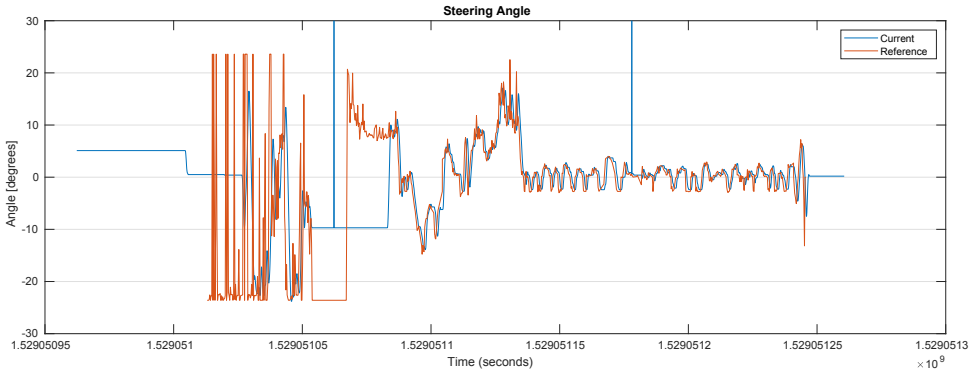


Fig. 6.70 Low level steering angle control performance with minimal configuration IV.

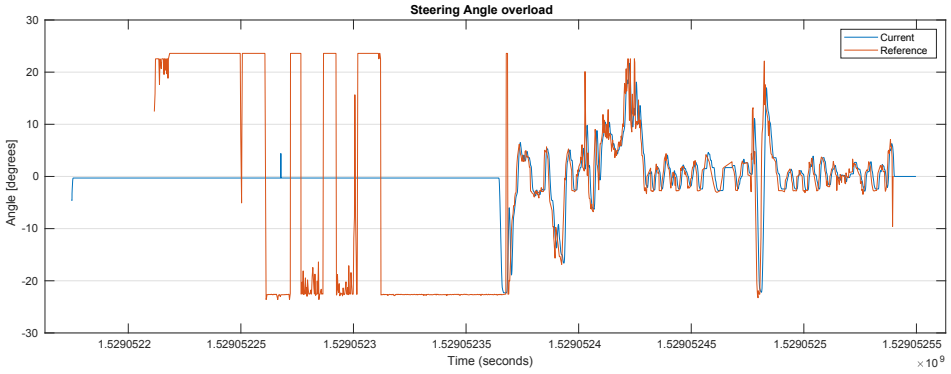


Fig. 6.71 Low level steering angle control performance with image processing configuration IV.

Chapter 7

Conclusions and Future work

7.1 Conclusions

The global conclusion is that the software architecture has accomplished all the requirements for self-driving navigation, and two research platforms with this architecture are fully functional, reliable and robust in the campus area of University Carlos III de Madrid.

One of the pillars of the navigation architecture is the connection of the four main modules, localization, mapping, control, and planning, that are optimized using the ROS framework. The shared messages between these modules are standard, providing more interchangeability with other modules and better understanding in the debugging stage.

The conclusion related to the global and local localization is that using the TF tool of the ROS framework is more valuable as doing by odometry messages due to the optimization of TF tool for this matter.

Moreover, it has been created two types of approach in the mapping module and both are completely valid. One, uses the costmap2D for global and local map fully integrated with the trajectory generation, and the other approach uses maps generated from scratch fully developed by LSI team which fuse all sensor data at high publish rate.

The planner module is based on a full test and wide openly shared tool called *move base* and fits very well with the software architecture proposed. This tool is ready to work with our own plug-ins of new and optimized trajectory generation algorithms.

This architecture has minimized the impact of the bottlenecks detected in the generation of

the costmap2D for local trajectory using the frontal laser sensor instead of the huge amount of data from lidar 360 degrees.

The low-level control of the vehicle has been fully developed using specific microcontrollers for the motors installed in the vehicles. The communication between this device and the software architecture is robust and failsafe due to the heartbeat checking between the device and the computer. Additionally, a brake mechanism has been mounted to guaranteed the correct behavior of the vehicle caused by unexpected events like the sudden apparition of pedestrians or dynamic obstacles.

The data acquired from the scenarios presented in the result section has been proved that the vehicle is able to navigate even with a high overloaded computer. The extra modules developed are intended for testing and research new methods for optimizing the main problems about self-driving navigation, hence, one of the most important parts of this architecture is the interchangeability between modules and the easy integration of new features to improve the overall performance.

The conclusions of the results obtained from the use cases demonstrate the good performance of the software architecture regarding the self-driving navigation in a structured environment. It has been shown in the three experiments how the vehicle navigates in the campus area avoiding collisions, recalculating the local trajectories for smoother paths and following the global trajectory.

This work has derived into a huge amount of knowledge and novelty ideas for self-driving vehicles, software integration and software architecture and this whole work has published 14 scientific publications listed in Appendix B.

7.2 Future Work

The iCab vehicles are electrical EZ-GO RXV 2008 and 2009 golf carts and the functional cycle are reaching the end due to the battery degradation and mechanical problems. For this reason, the vehicles will be replaced by Mitsubishi i-MiEV (Mitsubishi innovative electrical vehicle) in order to expand the navigation zones to roads with other vehicles. For this requirement, the architecture will suffer several changes such as the navigation based on GPS maps instead of the a priori known maps of the environment. New localization has to be developed whose main source will be the image processing due to the inaccuracy of the 16

planes lidar in high speed and minimal points to extract the features between two consecutive readings. The fusion and filter using UKF or EKF is a good approach. For the rest of the modules such as local costmap2D, local and global trajectory generation, it is necessary to replace the approach and use Bézier curves which are suitable and more coherence with the Ackermann model.

The lack of self-awareness state in the iCab project has been always a big drawback because this module is mandatory in order to improve safety around the vehicle (for the vehicle and pedestrians in the surroundings). The minimal self-awareness checking should be the sensor's heartbeats in order to avoid critical events where nodes are highly dependent on the sensor readings. Another check is the coherence of the commands generated as a reference and the performance of the vehicle. That means, when the reference is turned to the left and vehicle for mechanical problems (such as encoder error) and vehicle performance is highly different, send a warning to the self-awareness module to act accordingly. Furthermore, the vehicle requires a machine learning module to know when there is an abnormal behavior or inside of the abnormality, it learns to include as normal with repetition (learning from experience). For learning when an abnormal behavior should be included as normal from repetition, a work published by the author in collaboration of Mahdyar Ravanbakhsh, Mohamad Baydoun, Damian Campo, David Martin Lucio Marcenaro and Carlo Regazzoni in [75] shows the results obtained and the possibilities open to including in future architectures. The main idea of the self-awareness module is shown in figure 7.1 which contains the main self-awareness manager and the modules of self-awareness regarding the sensors, trajectory, movement, and perception. This self-awareness is intended to stop the vehicle in case of a critical emergency and create warnings to other modules in order to stay alert of the behavior of the vehicle in the event of abnormal activity. The final step is to integrate a learning module where the vehicle is able to distinguish different abnormalities and learn what abnormality is critical or replace the abnormality for normal behavior.

As a future work, it is necessary to link the MRTA module to the incoming goal point for navigation. This task is really simple, just a node to receive the port MRTA processing message and send the goal point to the desired vehicle. This node is already written and waiting to be tested in iCabs.

In the previous section has been talked about the Kinect 2 integration for backward movement and it is already tested. The step that is missing is the generation of the costmap2D with the frontal laser and the Kinect 2 occupancy grid map. The first approach is to use another layer in the costmap2D plugin which takes the grid map provided by Kinect 2 and fuse with the laser readings. This approach takes around 10 seconds to generate one useful costmap2D and needs to be improved in order to use it with the local trajectory.

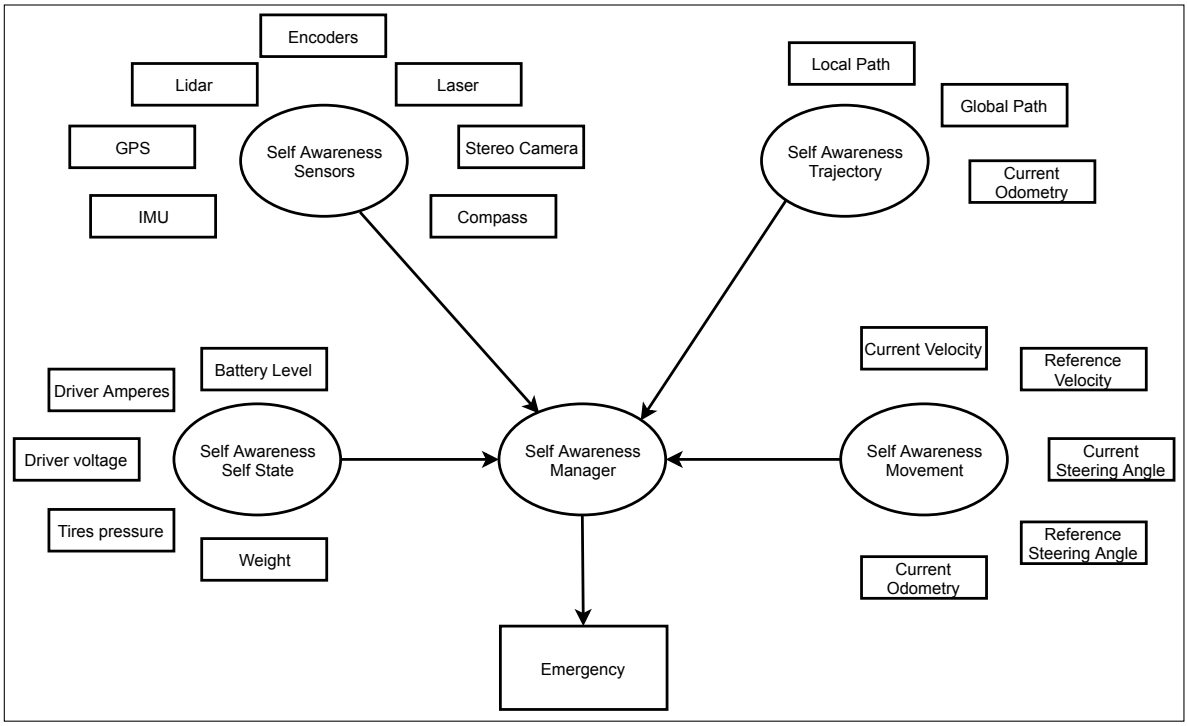


Fig. 7.1 Self-awareness architecture.

Appendix A

Configuration files

A.1 TF

In order to know the relations (distance) between the elements of the vehicle and the maps in order to know the intrinsic and extrinsic position of the sensors or just for transforming between global and local position, this project uses the tool TF (Transformation between Frames) that gather this information directly with a simple call of a function. First, in order to use this functionality, it is necessary to describe precisely the physical elements of the vehicle, such as the sensors, and the odometry reference. This elements are static and maintain the same position in the vehicle (see figures A.1 and A.2).

It is necessary to clarify the local coordinates where all frames will be related inside of the vehicle (\base_link) and the local coordinates where the vehicle will be related with the initial position (\odom). In order to debug and work better with the sensors and transformations, two extra frames have been added (\base_footprint and \rear_wheel_axis). \base_footprint is the reference to generate the local map with the sensors. This relation allows the composition of the gridmap in 2D over the floor (projection 2D). The relation second auxiliary frame \rear_wheel_axis is intended to work with the planners and the odometries. Due to the analytical composition of the encoder odometry, all other odometries has been referenced with properly transformations to the rear wheel axis, therefore, the planner checks the waypoints and the rear wheel axis in order to recalculate the waypoints or validate the arrival of the waypoint.

- rear_wheel_axis: this frame is the reference for the odometries and is located in the center of the rear wheel axis. It is located in $Pose[-0.5,0,0,0,0,0]$ from base footprint as the reference.
- base_footprint: this frame is the 2D base link projection. The use is to reference the sensors with the floor. $Pose[0,0,-0.4,0,0,0]$
- base_link: frame of reference(X,Y) of the vehicle. All sensors are referenced to this link. It is located in $Pose[0,0,0,0,0,0]$ as the main reference of the vehicle.
- lidar: physical location of the lidar sensor. $Pose[0.6201.41,0,0,0]$
- laser: physical location of the laser sensor. $Pose[1.1900.05,0,0,0]$
- gps: physical location of the gps sensor. $Pose[-0.3601.35,0,0,0]$

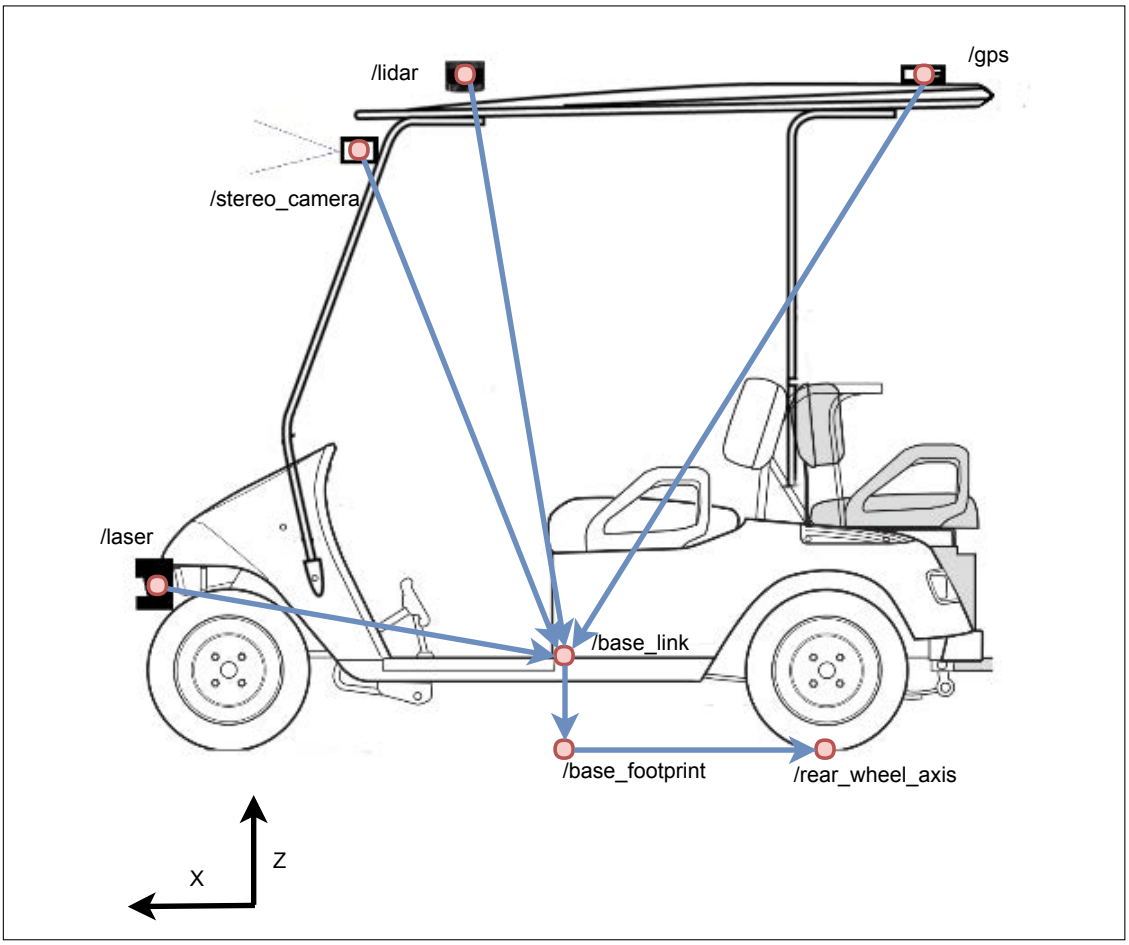


Fig. A.1 icab TF frames side view.

- stereo_camera: physical location of the stereo camera sensor. $Pose[0.8501.13, 0, 0, 0]$

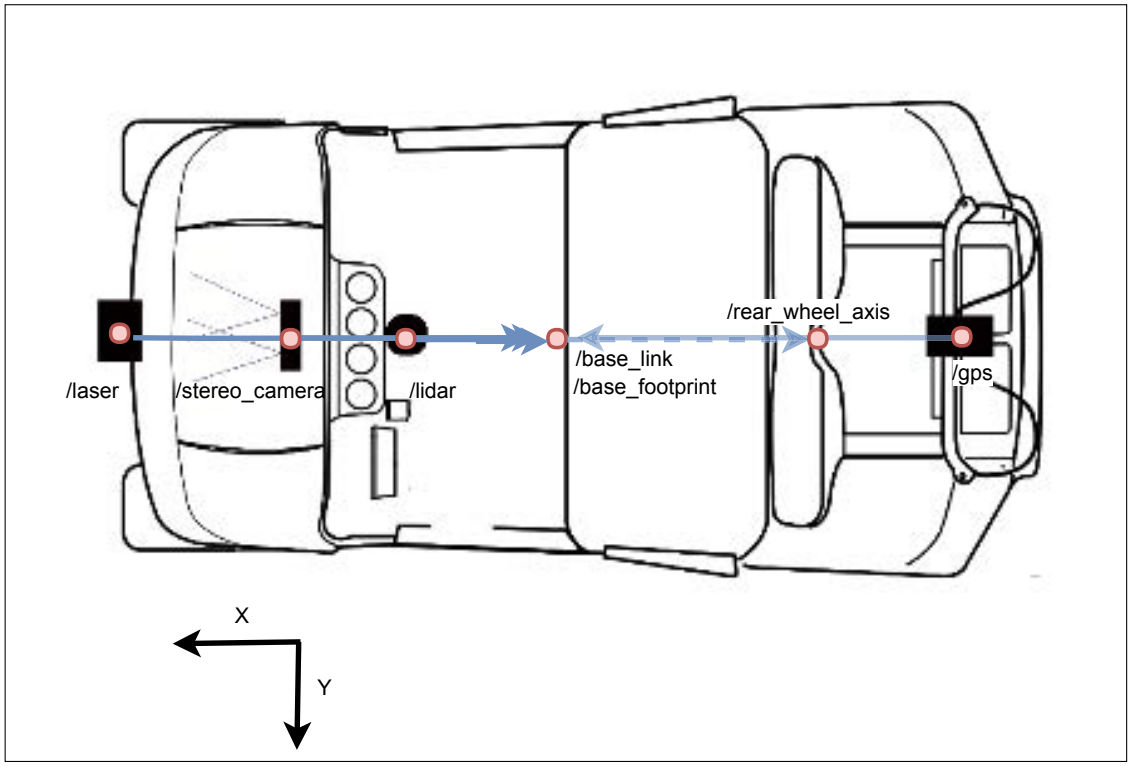


Fig. A.2 icab TF frames top view.

A.2 AMCL

```
use_map_topic: true

global_frame_id: "/map"

## Publish scans from best pose at a max of 10 Hz
odom_model_type: "diff"
odom_alpha5: 0.1
gui_publish_rate: 10.0
laser_max_beams: 60
laser_max_range: 12.0
min_particles: 500
max_particles: 2000
kld_err: 0.05
kld_z: 0.99
odom_alpha1: 0.2
odom_alpha2: 0.2
## translation std dev, m
odom_alpha3: 0.2
odom_alpha4: 0.2
laser_z_hit: 0.5
aser_z_short: 0.05
laser_z_max: 0.05
laser_z_rand: 0.5
laser_sigma_hit: 0.2
laser_lambda_short: 0.1
laser_model_type: "likelihood_field" # "likelihood_field" or "beam"
laser_likelihood_max_dist: 2.0
update_min_d: 0.25
update_min_a: 0.2

resample_interval: 1

## Increase tolerance because the computer can get quite busy
transform_tolerance: 1.0
recovery_alpha_slow: 0.001
recovery_alpha_fast: 0.1
```

Fig. A.3 AMCL configuration file.

A.3 Map Server

```
image: mapaUC3M_0_3_3.pgm
resolution: 0.300000
origin: [0, 0, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Fig. A.4 Map server configuration parameters.

A.4 CostMap2D

```
footprint: [[2, -0.6], [2, 0.6], [-0.5, 0.6], [-0.5, -0.6] ]

transform_tolerance: 0.2
map_type: costmap
```

Fig. A.5 Common parameters for local and global costmap.

```
global_costmap:
  global_frame: /map
  update_frequency: 1.0
  publish_frequency: 0.5
  static_map: true

  transform_tolerance: 0.5
  plugins:
    - {name: static_layer,            type: "costmap_2d::StaticLayer"}
    - {name: inflation_layer,         type: "costmap_2d::InflationLayer"}

  static_layer:
    enabled:              true
    map_topic:            "/map"
    subscribe_to_updates: true
  inflation_layer:
    enabled:              true
    cost_scaling_factor:  10.0
    inflation_radius:     0.5
```

Fig. A.6 Global Costmap2D parameters.

```
local_costmap:
  global_frame: /map
  update_frequency: 5.0
  publish_frequency: 4.0
  static_map: false
  rolling_window: true
  width: 30
  height: 30
  resolution: 0.7
  transform_tolerance: 0.5

  plugins:
   - {name: obstacle_layer,      type: "costmap_2d::ObstacleLayer"}
   - {name: inflation_layer,     type: "costmap_2d::InflationLayer"}

  obstacle_layer:
   enabled: true
   obstacle_range: 3.0
   raytrace_range: 3.5
   inflation_radius: 0.2
   track_unknown_space: false
   combination_method: 1
   min_obstacle_height: 0.5
   observation_sources: center_laser
   center_laser:
   {
      data_type: LaserScan,
      sensor_frame: $(arg vehicle)/laser,
      topic: scan,
      marking: true,
      clearing: true,
      obstacle_range: 10,
      raytrace_range: 10,
      min_obstacle_height: 0.1
   }
  inflation_layer:
    enabled:              true
    cost_scaling_factor:  10.0
    inflation_radius:     0.5
```

Fig. A.7 Local Costmap2D parameters.

Appendix B

Publications

- Marin-Plaza, P., Hussein, A., Martin, D., and Escalera, A. d. l. (2018b). Global and local path planning study in a ros-based research platform for autonomous vehicles. *Journal of Advanced Transportation*, 2018
- Hussein, A., Marín-Plaza, P., García, F., and Armingol, J. M. (2018). Hybrid optimization-based approach for multiple intelligent vehicles requests allocation. *Journal of Advanced Transportation*, 2018
- Marin-Plaza, P., Hussein, A., Gomez, D., and Escalera, A. (2018a). icab use case for ros-based architecture. *In Iberian Robotics (ROBOT2018)- Springer*, pages 1–31
- Ravanbakhsh, M., Baydoun, M., Campo, D., Marin, P., Martin, D., Marcenaro, L., and Regazzoni, C. S. (2018b). Learning multi-modal self-awareness models for autonomous vehicles from human driving. *arXiv preprint arXiv:1806.02609*
- Ravanbakhsh, M., Baydoun, M., Campo, D., Marin, P., Martin, D., Marcenaro, L., and Regazzoni, C. S. (2018a). Hierarchy of gans for learning embodied self-awareness model. *arXiv preprint arXiv:1806.04012*
- Baydoun, M., Ravanbakhsh, M., Campo, D., Marin, P., Martin, D., Marcenaro, L., Cavallaro, A., and Regazzoni, C. S. (2018). A multi-perspective approach to anomaly detection for self-aware embodied agents. *arXiv preprint arXiv:1803.06579*
- Hussein, A., Marin-Plaza, P., Garcia, F., and Armingol, J. (2017a). Optimization-based approach for cooperation and coordination of multi-autonomous vehicles. In *Sixteenth International Conference on Computer Aided Systems Theory (EUROCAST)*, pages 1–2
- Olier, J. S., Marín-Plaza, P., Martin, D., Marcenaro, L., Barakova, E., Rauterberg, M., and Regazzoni, C. (2017). Dynamic representations for autonomous driving. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE
- Hussein, A., Marín-Plaza, P., García, F., and Armingol, J. M. (2017b). Autonomous cooperative driving using v2x communications in off-road environment. In *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*, pages 1–6. IEEE

- Kokuti, A., Hussein, A., Marín-Plaza, P., de la Escalera, A., and García, F. (2017). V2x communications architecture for off-road autonomous vehicles. In *Vehicular Electronics and Safety (ICVES), 2017 IEEE International Conference on*, pages 69–74. IEEE

- Martin Gomez, D., Marin-Plaza, P., Hussein, A., Escalera, A., and Armingol, J. (2016). Ros-based architecture for autonomous intelligent campus automobile (icab). *UNED Plasencia Revista de Investigacion Universitaria*, 12:257–272

- Hussein, A., Marin-Plaza, P., Martin, D., de la Escalera, A., and Armingol, J. M. (2016b). Autonomous off-road navigation using stereo-vision and laser-rangefinder fusion for outdoor obstacles detection. *IEEE Intelligent Vehicles Symposium (IV)*, pages 104–109

- Marin-Plaza, P., Beltran, J., Hussein, A., Musleh, B., Martin, D., de la Escalera, A., and Armingol, J. M. (2016). Stereo vision-based local occupancy grid map for autonomous navigation in ros. *Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, 3:703–708

- Marín, P., Hussein, A., Martín, D., García, F., De la Escalera, A., and Armingol, J. (2016). Ros-based architecture for autonomous vehicles. *SEGVAUTO-TRIES-CM: Technologies for a Safe, Accessible and Sustainable Mobility*, pages 43–46

Bibliography

[1] Alessandrini, A., Cattivera, A., Holguin, C., and Stam, D. (2014). Citymobil2: challenges and opportunities of fully automated mobility. In *Road Vehicle Automation*, pages 169–184. Springer.

[2] Aurenhammer, F. (1991). Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405.

[3] Barron, J. L., Fleet, D. J., and Beauchemin, S. S. (1994). Performance of optical flow techniques. *International journal of computer vision*, 12(1):43–77.

[4] Baydoun, M., Ravanbakhsh, M., Campo, D., Marin, P., Martin, D., Marcenaro, L., Cavallaro, A., and Regazzoni, C. S. (2018). A multi-perspective approach to anomaly detection for self-aware embodied agents. *arXiv preprint arXiv:1803.06579*.

[5] Behere, S. and Torngren, M. (2015). A functional architecture for autonomous driving. In *Automotive Software Architecture (WASA), 2015 First International Workshop on*, pages 3–10. IEEE.

[6] Beiker, S. (2016). Deployment scenarios for vehicles with higher-order automation. In *Autonomous Driving*, pages 193–211. Springer.

[7] Beltran, J., Guindel, C., Moreno, F. M., Cruzado, D., Garcia, F., and de la Escalera, A. (2018). Birdnet: a 3d object detection framework from lidar information. *arXiv preprint arXiv:1805.01195*.

[8] Beltrán, J., Jaraquemada, C., Musleh, B., de la Escalera, A., and Armingol, J. M. (2017). Dense semantic stereo labelling architecture for in-campus navigation. In *VISIGRAPP (5: VISAPP)*, pages 266–273.

[9] Bertozzi, M., Bombini, L., Broggi, A., Buzzoni, M., Cardarelli, E., Cattani, S., Cerri, P., Debattisti, S., Fedriga, R., Felisa, M., et al. (2010). The vislab intercontinental autonomous challenge: 13,000 km, 3 months, no driver. In *Proc. 17th World Congress on ITS, Busan, South Korea*.

[10] Biswas, J. and Veloso, M. (2010). Wifi localization and navigation for autonomous indoor mobile robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4379–4384. IEEE.

[11] Broggi, A., Medici, P., Cardarelli, E., Cerri, P., Giacomazzo, A., and Finardi, N. (2010). Development of the control system for the vislab intercontinental autonomous challenge. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 635–640.

[12] Broggi, A., Medici, P., Zani, P., Coati, A., and Panciroli, M. (2012). Autonomous vehicles control in the vislab intercontinental autonomous challenge. *Annual Reviews in Control*, 36(1):161–171.

[13] Brozat, A. (2017). Sedric in shanghai: Self-driving car without a cockpit celebrates premiere in china.

[14] Buehler, M., Iagnemma, K., and Singh, S. (2007). *The 2005 DARPA grand challenge: the great robot race*, volume 36. Springer Science & Business Media.

[15] Canny, J. (1988). *The complexity of robot motion planning*. MIT press.

[16] CCAV (2017). *UK Connected and Autonomous Vehicle Research and Development Projects 2017*, volume 1.

[17] CDMV (2015). Summary of draft autonomous vehicles deployment regulations.

[18] Chong, K. S. and Kleeman, L. (1999). Feature-based mapping in real, large scale environments using an ultrasonic array. *The International Journal of Robotics Research*, 18(1):3–19.

[19] Cong, Y., Sawodny, O., Chen, H., Zimmermann, J., and Lutz, A. (2010). Motion planning for an autonomous vehicle driving on motorways by using flatness properties. In *Control Applications (CCA), 2010 IEEE International Conference on*, pages 908–913. IEEE.

[20] Darms, M. S., Rybski, P. E., Baker, C., and Urmson, C. (2009). Obstacle detection and tracking for the urban challenge. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):475–485.

[21] de la Escalera, A., Izquierdo, E., Martín, D., Musleh, B., García, F., and Armingol, J. M. (2016). Stereo visual odometry in urban environments based on detecting ground features. *Robotics and Autonomous Systems*, 80:1–10.

[22] Delsart, V., Fraichard, T., and Martinez-Gomez, L. (2009). Real-time trajectory generation for car-like vehicles navigating dynamic environments. In *IEEE Int. Conf. on Robotics and Automation*.

[23] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

[24] Everett, H. (1995). *Sensors for Mobile Robots: Theory and Applications.*, volume 1. Taylor & Francis.

[25] Ferris, B. D., Fox, D., and Lawrence, N. (2007). Wifi-slam using gaussian process latent variable models.

[26] Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999(343-349):2–2.

[27] Gibbs, S. (2017). What will the car of 2040 be like?

[28] Goodrich, M. A. (2002). Potential fields tutorial. *Class Notes*, 157.

[29] Guizzo, E. (2011). How google's self-driving car works. *IEEE Spectrum Online*, 18.

[30] Gutmann, J.-S., Burgard, W., Fox, D., and Konolige, K. (1998). An experimental comparison of localization methods. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 2, pages 736–743. IEEE.

[31] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

[32] Hernández, N., Herranz, F., Ocaña, M., Bergasa, L. M., Alonso, J. M., and Magdalena, L. (2009). Wifi localization system based on fuzzy logic to deal with signal variations. In *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1–6. IEEE.

[33] Higgins, T. (2017). The end of car ownership.

[34] Hillel, A. B., Lerner, R., Levi, D., and Raz, G. (2014). Recent progress in road and lane detection: a survey. *Machine vision and applications*, 25(3):727–745.

[35] Hong, S., Ko, H., and Kim, J. (2010). Vicp: Velocity updating iterative closest point algorithm. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1893–1898. IEEE.

[36] Hrbáček, J., Ripel, T., and Krejsa, J. (2010). Ackermann mobile robot chassis with independent rear wheel drives. In *Power Electronics and Motion Control Conference (EPE/PEMC), 2010 14th International*, pages T5–46. IEEE.

[37] Hussein, A. (2018). *Control and Communication Systems for Automated Vehicles Cooperation and Coordination*. PhD thesis, Universidad Carlos III de Madrid.

[38] Hussein, A., Garcia, F., Armingol, J. M., and Olaverri-Monreal, C. (2016a). P2v and v2p communication for pedestrian warning on the basis of autonomous vehicles. *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 2034–2039.

[39] Hussein, A., Marin-Plaza, P., Garcia, F., and Armingol, J. (2017a). Optimization-based approach for cooperation and coordination of multi-autonomous vehicles. In *Sixteenth International Conference on Computer Aided Systems Theory (EUROCAST)*, pages 1–2.

[40] Hussein, A., Marín-Plaza, P., García, F., and Armingol, J. M. (2017b). Autonomous cooperative driving using v2x communications in off-road environment. In *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*, pages 1–6. IEEE.

[41] Hussein, A., Marín-Plaza, P., García, F., and Armingol, J. M. (2018). Hybrid optimization-based approach for multiple intelligent vehicles requests allocation. *Journal of Advanced Transportation*, 2018.

[42] Hussein, A., Marin-Plaza, P., Martin, D., de la Escalera, A., and Armingol, J. M. (2016b). Autonomous off-road navigation using stereo-vision and laser-rangefinder fusion for outdoor obstacles detection. *IEEE Intelligent Vehicles Symposium (IV)*, pages 104–109.

[43] Ishiguro, H., Nishikawa, K., and Mor, H. (1992). Mobile robot navigation by visual sign patterns existing in outdoor environment. In *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on*, volume 1, pages 636–641. IEEE.

[44] Janai, J., Güney, F., Behl, A., and Geiger, A. (2017). Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *arXiv preprint arXiv:1704.05519*.

[45] Kato, S., Takeuchi, E., Ishiguro, Y., Ninomiya, Y., Takeda, K., and Hamada, T. (2015). An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68.

[46] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580.

[47] Kleiner, A., Prediger, J., and Nebel, B. (2006). Rfid technology-based exploration and slam for search and rescue. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4054–4059. IEEE conference proceedings.

[48] Kokuti, A., Hussein, A., Marín-Plaza, P., de la Escalera, A., and García, F. (2017). V2x communications architecture for off-road autonomous vehicles. In *Vehicular Electronics and Safety (ICVES), 2017 IEEE International Conference on*, pages 69–74. IEEE.

[49] Kuramachi, R., Ohsato, A., Sasaki, Y., and Mizoguchi, H. (2015). G-icp slam: An odometry-free 3d mapping system with robust 6dof pose estimation. In *Robotics and Biomimetics (ROBIO), 2015 IEEE International Conference on*, pages 176–181. IEEE.

[50] Labayrade, R., Aubert, D., and Tarel, J.-P. (2002). Real time obstacle detection in stereovision on non flat road geometry through" v-disparity" representation. In *Intelligent Vehicle Symposium, 2002. IEEE*, volume 2, pages 646–651. IEEE.

[51] Latecki, L. J. and Lakaemper, R. (2006). Polygonal approximation of laser range data based on perceptual grouping and em. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 790–796. IEEE.

[52] LaValle, S. M. and Kuffner Jr, J. J. (2000). Rapidly-exploring random trees: Progress and prospects.

[53] Lee, S. J., Cho, D. W., Chung, W. K., Lim, J. H., and Kang, C. U. (2005). Feature based map building using sparse sonar data. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 1648–1652. IEEE.

[54] Leonard, J. J. and Durrant-Whyte, H. F. (1991a). Mobile robot localization by tracking geometric beacons. *IEEE Transactions on robotics and Automation*, 7(3):376–382.

[55] Leonard, J. J. and Durrant-Whyte, H. F. (1991b). Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee.

[56] Luis, H. A. (2018). *iCab web server*. PhD thesis, Universidad Carlos III de Madrid.

[57] Ma, L., Yang, J., and Zhang, M. (2012). A two-level path planning method for on-road autonomous driving. In *Intelligent System Design and Engineering Application (ISDEA), 2012 Second International Conference on*, pages 661–664. IEEE.

[58] Marín, P., Hussein, A., Martín, D., García, F., De la Escalera, A., and Armingol, J. (2016). Ros-based architecture for autonomous vehicles. *SEGVAUTO-TRIES-CM: Technologies for a Safe, Accessible and Sustainable Mobility*, pages 43–46.

[59] Marin-Plaza, P., Beltran, J., Hussein, A., Musleh, B., Martin, D., de la Escalera, A., and Armingol, J. M. (2016). Stereo vision-based local occupancy grid map for autonomous navigation in ros. *Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, 3:703–708.

[60] Marin-Plaza, P., Hussein, A., Gomez, D., and Escalera, A. (2018a). icab use case for ros-based architecture. *In Iberian Robotics (ROBOT2018)- Springer*, pages 1–31.

[61] Marin-Plaza, P., Hussein, A., Martin, D., and Escalera, A. d. l. (2018b). Global and local path planning study in a ros-based research platform for autonomous vehicles. *Journal of Advanced Transportation*, 2018.

[62] Martin Gomez, D., Marin-Plaza, P., Hussein, A., Escalera, A., and Armingol, J. (2016). Ros-based architecture for autonomous intelligent campus automobile (icab). *UNED Plasencia Revista de Investigacion Universitaria*, 12:257–272.

[63] Masehian, E. and Sedighizadeh, D. (2007). Classic and heuristic approaches in robot motion planning-a chronological review. *World Academy of Science, Engineering and Technology*, 23:101–106.

[64] Matias, B., Oliveira, H., Almeida, J., Dias, A., Ferreira, H., Martins, A., and Silva, E. (2015). High-accuracy low-cost rtk-gps for an unmannned surface vehicle. In *OCEANS 2015-Genova*, pages 1–4. IEEE.

[65] Musleh, B., de la Escalera, A., and Armingol, J. M. (2011). Uv disparity analysis in urban environments. In *International Conference on Computer Aided Systems Theory*, pages 426–432. Springer.

[66] Musleh, B., Martín, D., Armingol, J. M., and de la Escalera, A. (2014). Continuous pose estimation for stereo vision based on uv disparity applied to visual odometry in urban environments. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3983–3988. IEEE.

[67] Musleh, B., Martin, D., de la Escalera, A., and Armingol, J. M. (2012). Visual ego motion estimation in urban environments based on uv disparity. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 444–449. IEEE.

[68] Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. Ieee.

[69] Nourani-Vatani, N., Roberts, J., and Srinivasan, M. V. (2009). Practical visual odometry for car-like vehicles. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3551–3557. IEEE.

[70] Olier, J. S., Marín-Plaza, P., Martin, D., Marcenaro, L., Barakova, E., Rauterberg, M., and Regazzoni, C. (2017). Dynamic representations for autonomous driving. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE.

[71] Organization, W. H. (2015). *Global status report on road safety 2015*. World Health Organization.

[72] Osman, M., Hussein, A., Al-Kaff, A., and García, F. (2018). Online adaptive covariance estimation approach for multiple odometry sensors fusion. *IEEE Intelligent Vehicles Symposium*.

[73] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3. Kobe.

[74] Ravanbakhsh, M., Baydoun, M., Campo, D., Marin, P., Martin, D., Marcenaro, L., and Regazzoni, C. S. (2018a). Hierarchy of gans for learning embodied self-awareness model. *arXiv preprint arXiv:1806.04012*.

[75] Ravanbakhsh, M., Baydoun, M., Campo, D., Marin, P., Martin, D., Marcenaro, L., and Regazzoni, C. S. (2018b). Learning multi-modal self-awareness models for autonomous vehicles from human driving. *arXiv preprint arXiv:1806.02609*.

[76] Resende, P., Nashashibi, F., Charlot, F., Holguin, C., Bouraoui, L., and Parent, M. (2012). A cooperative personal automated transport system-a citymobil demonstration in rocquencourt. In *IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE.

[77] Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F., and Bertram, T. (2012). Trajectory modification considering dynamic constraints of autonomous robots. In *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, pages 1–6. VDE.

[78] Rosmann, C., Feiten, W., Wosch, T., Hoffmann, F., and Bertram, T. (2013). Efficient trajectory optimization using a sparse model. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 138–143. IEEE.

[79] RTMaps (2018). Rtmaps capabilities.

[80] Ruppelt, J. and Trommer, G. F. (2016). Stereo-camera visual odometry for outdoor areas and in dark indoor environments. *IEEE Aerospace and Electronic Systems Magazine*, 31(11):4–12.

[81] SAE (2014). Automated driving - levels of driving automation. *SAE International*, 1(J3016).

[82] Scaramuzza, D. and Fraundorfer, F. (2011). Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92.

[83] Subramanian, A. P., Deshpande, P., Gao, J., and Das, S. R. (2008). Drive-by localization of roadside wifi networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 718–725. IEEE.

[84] Sun, H., Deng, W., Zhang, S., Wang, S., and Zhang, Y. (2014). Trajectory planning for vehicle autonomous driving with uncertainties. In *Informative and Cybernetics for Computational Social Systems (ICCSS), 2014 International Conference on*, pages 34–38. IEEE.

[85] Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

[86] Thrun, S. (2002). Probabilistic robotics. *Communications of the ACM*, 45(3):52–57.

[87] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. MIT press.

[88] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al. (2006). Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692.

[89] Ulrich, I. and Nourbakhsh, I. (2000). Appearance-based obstacle detection with monocular color vision. In *AAAI/IAAI*, pages 866–871.

[90] Urmson, C., Anhalt, J., Clark, M., Galatali, T., Gonzalez, J. P., Gowdy, J., Gutierrez, A., Harbaugh, S., Johnson-Roberson, M., Kato, H., et al. (2004). High speed navigation of unrehearsed terrain: Red team technology for grand challenge 2004. *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-04-37*.

[91] Wang, M., Ganjineh, T., and Rojas, R. (2011). Action annotated trajectory generation for autonomous maneuvers on structured road networks. In *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, pages 67–72. IEEE.

[92] Xiao, J. (2002). Introduction to robotics. *Robot Motion Planning, Department of Electronic Engineering City College of New York*.

[93] Xu, P., Dherbomez, G., Héry, E., Abidli, A., and Bonnifait, P. (2018). System architecture of a driverless electric car in the grand cooperative driving challenge. *IEEE Intelligent Transportation Systems Magazine Special*, 10(1):47–59.

[94] Yan, M., Wang, J., Li, J., and Zhang, C. (2017). Loose coupling visual-lidar odometry by combining viso2 and loam. In *Control Conference (CCC), 2017 36th Chinese*, pages 6841–6846. IEEE.

[95] YARP (2018). What exactly is yarp?

[96] Zhang, J. and Singh, S. (2014). Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2.

[97] Zhang, J. and Singh, S. (2015). Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2174–2181. IEEE.

[98] Ziegler, J., Bender, P., Schreiber, M., Lategahn, H., Strauss, T., Stiller, C., Dang, T., Franke, U., Appenrodt, N., Keller, C. G., et al. (2014). Making bertha drive-an autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, 6(2):8–20.